Predictive Articulatory speech synthesis Utilizing Lexical Embeddings (PAULE)

Dissertation
der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Konstantin Florian Sering
aus Berlin

Tübingen
2023

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	16.10.2023
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter/-in:	Prof. Dr. R. Harald Baayen
2. Berichterstatter/-in:	Prof. Dr. Martin V. Butz
3. Berichterstatter/-in:	Prof. Dr. Bernd Möbius

Zusammenfassung

Das Predictive Articulatory speech synthesis Utilizing Lexical Embeddings (PAULE) Modell ist ein neues Modell zur Kontrolle des artikulatorischen Sprachsynthesizers VocalTractLab (VTL)[15]. Mit PAULE lassen sich deutsche Wörter synthetisieren. Die Wortsynthese kann entweder mit Hilfe eines semantischen Vektors, der die Wortbedeutung kodiert, und der gewünschten Dauer der Wortsynthese gestartet werden oder es kann eine Resynthese von einer Audiodatei gemacht werden. Die Audiodatei kann beliebige Aufnahmen von Sprecher:innen enthalten, wobei die Resynthese immer über den Standardsprecher des VTL erfolgt. Abhängig von der Wortbedeutung und der Audiodatei variiert die Synthesequalität.

Neu an PAULE ist, dass es einen prädiktiven Ansatz verwendet, indem es aus der geplanten Artikulation die dazugehörige perzeptuelle Akustik vorhersagt und daraus die Wortbedeutung ableitet. Sowohl die Akustik als auch die Wortbedeutung sind als metrische Vektorräume implementiert. Dadurch lässt sich ein Fehler zu einer gewünschten Zielakustik und Zielbedeutung berechnen und minimieren. Bei dem minimierten Fehler handelt es sich nicht um den tatsächlichen Fehler, der aus der Synthese mit dem VTL entsteht, sondern um den Fehler, der aus den Vorhersagen eines prädiktiven Modells generiert wird. Obwohl es nicht der tatsächliche Fehler ist, kann dieser Fehler genutzt werden, um die tatsächliche Artikulation zu verbessern. Um das prädiktive Modell mit der tatsächlichen Akustik in Einklang zu bringen, hört sich PAULE selbst zu.

Ein in der Sprachsynthese zentrales Eins-Zu-Viele-Problem ist, dass eine Akustik durch viele verschiedene Artikulationen erzeugt werden kann. Dieses Eins-Zu-Viele-Problem wird durch die Vorhersagefehlerminimierung in PAULE aufgelöst, zusammen mit der Bedingung, dass die Artikulation möglichst stationär und mit möglichst konstanter Kraft ausgeführt wird. PAULE funktioniert ohne jegliche symbolische Repräsentation in der Akustik (Phoneme) und in der Artikulation (motorische Gesten oder Ziele). Damit zeigt PAULE, dass sich gesprochene Wörter ohne symbolische Beschreibungsebene modellieren lassen. Der gesprochenen Sprache könnte daher im Vergleich zur geschriebenen Sprache eine fundamental andere Verarbeitungsebene zugrunde liegen. PAULE integriert Erfahrungswissen sukzessive. Damit findet PAULE nicht die global beste Artikulation sondern lokal gute Artikulationen. Intern setzt PAULE auf künstliche neuronale Netze und die damit verbundenen Gradienten, die zur Fehlerkorrektur verwendet werden.

PAULE kann weder ganze Sätze synthetisieren noch wird somatosensorisches Feedback berücksichtigt. Zu Beidem gibt es Vorarbeiten, die in zukünftige Versionen integriert werden sollen.

Abstract

The Predictive Articulatory speech synthesis Utilizing Lexical Embeddings (PAULE) model is a new control model for the VocalTractLab (VTL)[15] speech synthesizer, a simulator of the human speech system. It is capable of synthesizing single words in the German language. The speech synthesis can be based on a target semantic vector or on target acoustics, i.e., a recorded word token. VTL is controlled by 30 parameters. These parameters have to be estimated for each time point during the production of a word, which is roughly every 2.5 milliseconds. The time-series of these 30 control parameters (cps) of the VTL are the control parameter trajectories (cp-trajectories). The high dimensionality of the cp-trajectories in combination with non-linear interactions leads to a many-to-one mapping problem, where many sets of cp-trajectories produce highly similar synthesized audio.

PAULE solves this many-to-one mapping problem by anticipating the effects of cp-trajectories and minimizing a semantic and acoustic error between this anticipation and a targeted meaning and acoustics. The quality of the anticipation is improved by an outer loop, where PAULE listens to itself. PAULE has three central design features that distinguish it from other control models: First, PAULE does not use any symbolic units, neither motor primitives, articulatory targets, or gestural scores on the movement side, nor any phone or syllable representation on the acoustic side. Second, PAULE is a learning model that accumulates experience with articulated words. As a consequence, PAULE will not find a global optimum for the inverse kinematic optimization task it has to solve. Instead, it finds a local optimum that is conditioned on its past experience. Third, PAULE uses gradient-based internal prediction errors of a predictive forward model to plan cp-trajectories for a given semantic or acoustic target. Thus, PAULE is an error-driven model that takes its previous experiences into account.

Pilot study results indicate that PAULE is able to minimize an acoustic semantic and acoustic error in the resynthesized audio. This allows PAULE to find cp-trajectories that are correctly classified by a classification model as the correct word with an accuracy of 60 %, which is close to the accuracy for human recordings of 63 %. Furthermore, PAULE seems to model vowel-to-vowel anticipatory coarticulation in terms of formant shifts correctly and can be compared to human electromagnetic articulography (EMA) recordings in a straightforward way. Furthermore, with PAULE it is possible to condition on already executed past cp-trajectories and to smoothly continue the cp-trajectories from the current state. As a side-effect of developing PAULE, it is possible to create large amounts of training data for the VTL through an automated segment-based approach.

Next steps, in the development of PAULE, include adding a somatosensory feedback channel, extending PAULE from producing single words to the articulation of small utterances and adding a thorough evaluation.

Acknowledgments

My main line of research, which this thesis summarizes, evolved over the last seven years. In creating the Predictive Articulatory speech synthesis Utilizing Lexical Embeddings (PAULE) model and in finding appropriate individual LSTM-based models, Paul Schmidt-Barbo did an excellent job and I am very thankful and indebted to him. Thank you, Paul! Besides working with me and for me as a student assistant, he added the ability to start directly from semantic vectors to the PAULE model in his master thesis. When creating PAULE would be seen like building a house, I am the architect of the PAULE model and Paul Schmidt-Barbo is the foreman. The model and my thesis would not have the same quality without him.

In the picture of building a house, my two main supervisors, R. Harald Baayen and Martin V. Butz, are the building contractors that supplied me with enough time and money to work on the house and give guidance whenever general design decisions were needed. I am very happy and thankful that they allowed me to do things my way and that they supported me over the full time of my research. Especially, in the beginning, when I was still completely new to the field, they helped me to find my research niche and connected me to collaborators in Frankfurt am Main and Edmonton.

My third supervisor Benjamin V. Tucker is the stress analyst of the house. He warned me about the complexity of my self-selected task on the one hand and gave me valuable advice on design decisions on the other hand. Additionally, Benjamin Tucker supplied me with plenty of resources where I only used a small fraction in the end. Thank you Ben for your support and that you were always available if the need arose.

My collaborators helped me as craftspeople to build and focus on different aspects of the PAULE model and building PAULE without them would lack important aspects. They are: Fabian Tomaschek, Sebastian Otte, Elnaz Shafaei-Bejestan, Petar Milin, and Motoki Saito. I have to thank the secretary and lab manager Tineke Oushoorn, who helped me with the paperwork that is needed to do research at a German university.

Thanks go to my colleagues in the Quantitative Linguistics and the Cognitive Modeling groups, the organizers and participants of the ESSV conferences, and the Machine Learning Cluster in Tübingen. All these people and institutions gave me a supportive and critical surrounding, from which my research benefited a lot. Whenever I wanted it, I found some researchers who listened to my scientific problems and the ideas on how to solve them and supplied feedback or discussed the topic with me. Even if there is no list of names here, I am very thankful to this not explicitly mentioned group of people and the individual persons in it.

Special thanks go to Peter Birkholz, Simon Stone, and Yingming Gao from TU Dresden, who developed the VocalTractLab (VTL) and supplied me with different versions of the code and always were supportive and helpful when questions arose.

Thanks go to my excellent students that did projects with me or wrote a thesis under my supervision. They worked on different aspects of the PAULE model or closely connected topics. In chronological order they are: David-Elias Künstle, Marc Weitz, Niels Stehwien, Hannah Schütt, Tobias Menge, Maïwenn Fleig, and Paul Schmidt-Barbo.

Thanks go to Michael Gschwender and Nora Wickelmaier for reading and correcting my thesis. Both gave me valuable feedback on how to improve the structure of the thesis and improved the readability, reduced the number of spelling errors, and pointed out grammatical errors and incomplete sentences.

Finally, I thank my wife Nina and my three children Ella, Enno, Leo for being around and supportive and always opening new perspectives on this fascinating world.

Financial support

This research was financially supported in parts by the state Baden-Württemberg through the University Tübingen and the European Union through an ERC Advanced Grant (no. 742545) awarded to R. Harald Baayen. At the time of writing, I am a member of the Machine Learning Cluster of Excellence, EXC number 2064/1 – Project number 390727645.

Preface

After finishing my Diploma in Physics and my Diploma in Psychology, I joined R. Harald Baayen's Quantitative Linguistics group (Quantling) in 2015. I started with three months of getting to know the Quantling group and learning what research was conducted here. After this initial phase, Harald and I decided that my research should focus on the modeling of human speech production. At that time, the symbolic foundation of language was questioned and gradual, embedding, and learning-based methods were applied to different aspects of human language perception and production. Different learning algorithms were able to model psycholinguistic findings without the use of symbolic atomic units like graphemes or phonemes. In the Quantling group, human learning was modeled by predicting outcomes from cues. Outcomes and cues both could be present or absent in a learning event and the predictive value of a cue to an outcome was summarized in a weight matrix. These weights were updated using the Rescorla-Wagner learning rule. While being present or absent, the cues and outcomes still had some binary and symbolic notion. Still, the cues and outcomes used were qualitatively different from classical symbolic units like graphemes and phonemes.

New to the field of linguistics and with a methods, programming, and statistics background, I learned a lot and supported different researchers in different tasks in the first years. I took over the maintenance and responsibility of the compute hardware (servers, network attached storage, networking) and for the computers my colleagues in the Quantling group were using. Furthermore, I became the maintainer of the two R packages `ndl`[5] and `ndl2`[100], which were used to learn the weights in the learning models that model human language. Later, I reimplemented and improved the fast learning algorithm of `ndl2` in Python as the `pyndl` package[97], which allowed for better utf-8 support and got rid of a size limitation on the size of the weights. Furthermore, `pyndl` allowed to implement generalized versions of the Rescorla-Wagner learning rule and allowed for gradual cues and outcomes. As teaching duties, I took over the statistics course, which is the only course where general linguists are exposed to quantitative statistical methods and where at the same time computer linguists should learn their basics in statistics.

As my background in Physics and Psychology as well as the planned Ph.D. research were perfectly matching a Cognitive Science track, I joined Martin V. Butz's Cognitive Modeling group and registered my Ph.D. in Cognitive Science in the science faculty. This came with the added benefit that regulations from the science faculty applied to me, which I already knew from my studies in Psychology and Physics.

As someone joining without any linguistics training, but with knowledge in Psychology, Physics, and Computer Science, I neither wanted to believe everything new I learned about language and speech production nor question everything fundamentally. Especially,

I wanted to withstand the urge of the physicist in me to know everything better. I attended some courses and read some books, but my main way of getting into the field was by supporting different research with my statistics and methods skills. This allowed me to get some in-depth knowledge of the research without the need for substantial domain knowledge. Nevertheless, my first attempts at modeling speech production with an evolutionary algorithm failed. This failure was due to the huge size and high dimensionality of the motor space, problems in defining a proper scoring function, and my eagerness to include time sequences of variably length from the beginning.

Throughout the whole seven-year period, I balanced my work duties as a researcher and scientist with my family duties as a father of now three children and a full-time working wife. This balancing act was made possible by working only part-time between 20 to 24 hours a week and the possibility of my wife to work on Saturdays. As we have no grandparents or uncles around, childcare was a challenge, especially in Corona and during lock-down times. Due to my wife's excellent organizational skills, we managed to balance things out most of the time. In the end, I am thankful to my wife and happy and proud of the result I achieved in the last seven years as a half-time scientist. The downside of being a half-time scientist is that it is nearly never possible to fully dive into a scientific topic and separate oneself from the outside world. The upside is that every thought, scientific problem, and possible solution has more time to sink in and that a healthy distance is kept to one's research. These forces reflected decisions and prioritizations about the next research steps.

In 2016, after attending the Tagung experimentell arbeitender Psychologen (TeaP)[3,30], I attended a small and private workshop at the Frankfurt Institute for Advanced Studies (FIAS) in Frankfurt am Main, Germany, about goal-directed speech production with echo-state networks and the VocalTractLab (VTL) articulatory speech synthesizer[15] by Peter Birkholz. Peter Birkholz is a very kind guy from TU Dresden. He shared the code for the VTL with me and we developed some tooling together at the workshop to run the VTL API on Linux and interface it with Python. This workshop directed my research into creating a control model for the VTL. The workshop in Frankfurt was followed by a lab visit in Edmonton, Alberta Canada, with the phonetician Benjamin V. Tucker, who became my third supervisor besides R. Harald Baayen and Martin V. Butz. I learned more about speech and speech processing and presented my first ideas for my Ph.D. program for the first time to an audience that had in-depth knowledge of speech perception and speech production[85]. The year 2016 was completed by a joint measurement of Schlieren photography in front of the mouth and nose at the Deutsches Zentrum für Luft- und Raumfahrt (DLR) in Lampoldshausen, which resulted in a small corpus[107].

After attending the Machine Learning Summer School in 2017, I decided to stop pursuing the evolutionary algorithm to control the VTL and produce intelligible speech and started working on an idea of a gradient-aware planning algorithm with Martin V. Butz. During this time, the acoustic representation was still very much the one used in[4], which showed good performances in perception but seemed to lack important detail for speech production. I only realized in the upcoming years that the representation in[4] lacks too much detail for speech production.

In 2018, Fabian Tomaschek and I did a joint measurement with colleagues from the Max-Planck-Institut (MPI) Intelligent Systems in Tübingen and the Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI) in Saarbrücken, where we measured electromagnetic articulography (EMA) and 3-dimensional dynamic surface scans of the head [102]. Later this year, I looked at a multi-label problem [90] together with R. Harald Baayen and Petar Milin. For this research, the reimplementation of the learning algorithm in Python with `pyndl` came in handy. Furthermore, this research supported investigations into applying linear algebra and the delta-rule onto lexical embeddings and language perception and production and resulted in the Linear Discriminative Learning (LDL) model [7].

From 2019 onwards, I attended and presented the progress of my research at the Konferenz zur elektronischen Sprachsignalverarbeitung (ESSV). My first contribution was an automated method to generate large amounts of training data [94] for the VTL. My research made progress and gradient-aware planning ideas were presented in different contexts [86,87]. From 2020 to 2022 my project and progress to an integrated model of articulatory speech production that uses lexical embeddings for its gradient-aware planning, were refined. Pilot studies were conducted and presented at the ESSV and International Seminar on Speech Production (ISSP) conference meetings [95,93,82,96,81,91,92,88,89]. In this period, some of my collaborations that are not related to my Ph.D. research came to an end as well [108,31,10,107,8].

That Paul Schmidt-Barbo joined me in 2020 was very helpful for my research. Furthermore, I am happy that I had excellent students to supervise from the cognitive science and machine learning programs and that enough financial and computational resources had been at hand at all time. Due to funding through the European Union, I stopped teaching in 2018, which helped to focus my working hours as a half-time scientist on my research. My supportive role was still active by maintaining the computing infrastructure of the Quantling group and by maintaining several software packages. This thesis will introduce the Predictive Articulatory speech synthesis Utilizing Lexical Embeddings (PAULE) model, the result of the collaborative work that I pursued in the last years as a half-time scientist and which resulted in a fully-functioning cognitively motivated and plausible speech production model with a high-dimensional articulatory speech synthesizer, which is directly informed by the predicted meaning of the planned speech.

Contents

1 Introduction

> Derivation of meaning is hence the ultimate goal of language processing — and meaning is the start of the production process.
>
> Harley, 2013 [44]

Spoken language and spoken communication are a central part of human society, culture and the human communicative system in general. Babies acquire the skills of understanding short utterances in their mother tongue in the first year of life. Producing intelligible and meaningful utterances can take as long as three to four years in normal development and improves along the whole life span. [83,13,45]

The ability to speak, once acquired seems so natural and effortless to most of us that it is, at first glance, surprising that it is still difficult to mimic or synthesize speech with mechanical devices or computer-based simulators [110,66,104]. Huge improvements have been made in the last couple of years with the application of machine learning techniques and deep learning combined with signal processing to understand (written) language better and use it to create a good synthesis of human speech [114]. But there is still some way to go until machines will be able to speak and to communicate as naturally as humans do.

When it comes to our understanding of the processes concerning language, most of our knowledge stems from studies that use written language as their primary source of information. This concerns, for example, our understanding of semantics [58], morphology [74] and even phonology [23]. The basic skills to read and write are only acquired by most humans by the age of 5 years to 10 years – and in some case, many language users never master this skill in their whole life span. Around 5% to 10% of the human population even in the most developed countries never fully master basic reading and writing skills. In 2018, a survey estimated that 12 % of the German adult population can only read and write at the word-level at best and cannot read or write full sentences or texts [19,42]. This is in stark contrast to spoken language, which almost all adults are proficient in. Furthermore, over the timeline of mankind, the cultural skill to read and write and the use of reading and writing for a substantial part of daily communication is rather young. Before using reading and writing to communicate on a daily basis, the main use of written language were financial transactions, documenting laws, recording myths and historical events, and communication through letters [32].

Many languages differ systematically and substantially between their written and their spoken form [22,16,78]. Still, written and spoken forms have enough in common to classify them as belonging to the same language. The variability between different speakers and their variability over time is a lot higher in spoken language compared

to writing, for which strict orthographic norms are enforced (see[41,49,59,116] for speech and[11,12] for orthography). Furthermore, changes to the writing system and in written language are slower compared to systematic changes in spoken language[46,9]. This high variability and change in spoken language let to the argument, that speech has to be seen as something separate to language as it is too transient in its nature. It is claimed by[25] that speech is an imperfect realisation of the underlying true formal language system. In this thesis we will be more inclusive and count speech as part of language.

It is only in the informal contexts of texting that the natural variability in language use re-emerges. This raises the question of why it is that spoken language is easier to learn, more efficient to produce, and more prevalent in human communication, if it is at the same time more variable, less predictable, and faster changing than written language.

A large body of current language research focuses on written language either by creating Deep learning models for *natural language understanding* like GPT-3[18] or translation models such as google translate or deepl, which are both trained on texts , not on speech. In psycholinguistics, a substantial amount of research is conducted on the visual modality (reading) and the hand motion and finger fine control in writing tasks. Given the prevalence of human spoken communication in our daily lives, both the acoustic domain (listening) and the production of speech with the human articulatory system are underrepresented.

One of the reasons for the lack of prevalent speech and articulatory modeling research is the difficulty of doing quantitative, replicable research on it. There is on the one hand the transient nature of human speech[76].

Where a written word is easily archived and read over and over again, the human speech signal is gone, if not recorded at the moment of speaking. On the other hand, working with speech data was extremely time-consuming before we had speech editors along with the computational power to process it; researchers had to work with tape recordings, and "splicing" was done with tapes[26]. Still, there are branches in linguistics, in engineering, in communication research, and in psychology that focus on the domains of perceiving and producing spoken language and speech. This research has created a few big data sets, like the MALD[112] and the Redhen[51] data sets. These branches have been around for at least 100 years but faced until recently some big methodological challenges.

The big challenges in this field have been the large amounts of storage needed to store decent amounts of speech recordings; the computation power needed to crunch through the data once it has been collected; the high variability in the data; inverse problems like decomposing recordings that mix different speakers into one signal or remove included ambient noise; and the lack of fully automatic tools to extract and process big amounts of useful features from spoken language add to this challenge. Even now, a lot of handcrafting and fine-tuning is needed for rather simple tasks such as extracting pitches, aligning subtitles at the word level, and adding phonetic segment transcriptions to an audio signal[105].

But there is progress, many of the obstacles that have been posing problems in the field of speech technologies and spoken language processing have been solved technically

or mitigated recently. Storage has become a lot cheaper and easily available in the last decade, and with the rise of supercomputers and graphics cards, computations that used to take huge amounts of energy and time can now be conducted on consumer hardware. Furthermore, technological advances in lossy audio codecs (like mp3 or Vorbis) [1] but also in building cochlea implants, improved our knowledge of how human auditory perception and speech perception works. Therefore, it might be the right time to come back to the old problem of building a machine that produces speech similar to how humans produce their speech.

The focus of this thesis is on the investigation of the interaction of the human speech system with its physical limitations and constraints and the learned behavior of controlling and using this speech system to produce intelligible, natural speech. The control of the speech system is inspired by models of motor control in arm and hand movements and is developed along ten guiding principles (see next section). These guiding principles assume that speech is acquired by humans in the same way as other motor skills. The work in this thesis therefore can be seen as a proof of concept that the control of the human speech system does not necessarily require any language specific module or rule based symbolic representations. This approach is in stark contrast to most linguistic approaches of this task [25,110,69]. The findings in this thesis therefore suggest that speech at the word level does not necessary need any symbolic internal representations nor any rules to concatenate symbolic units like motor gestures or phonemes.

1.1 Guiding principles

The framework of speech production that developed in this thesis and named Predictive Articulatory speech synthesis Utilizing Lexical Embeddings (PAULE) is based on the following ten guiding principles. After the list of principles each of the guiding principles is shortly described, setting the scene for the rest of this thesis. The overall rationale behind PAULE is coded and reflected in these principles.

The guiding principles are:

1. The word is the smallest meaningful unit
2. Meaning is the start and end of speech production
3. Speech production is goal-directed
4. Speech production is adaptive and error-driven
5. Speech is learned behavior
6. Articulation is influenced by experience

[1] As lossy audio codecs aim for the best compression, i.e. lowest information density or bit rate, while still being intelligible or even encoding all acoustically perceptual information, the way the encoding and decoding works, can tell us something about the human auditory system. Furthermore, the bit rate of the audio codec can be used as an upper bound for information content in the signal.

7. Sublexical units like the phoneme are not necessary (no phonemes)
8. Gestural scores or movement targets are not necessary (no gestures)
9. The model should be as simple as possible, while still being able to model some main features of human behavior
10. Using time-series to time-series and time-series to fixed-vector models

1.1.1 The word is the smallest meaningful unit

In this thesis, we assume that words are the smallest stand-alone meaningful unit. Presenting a word in isolation conveys some meaning to the listener. Sub-word units like morphemes or even single sounds in minimal pairs, like /pin/ and /bin/, are important features of these words, which to some extent encode, through their systematic use over the vocabulary in a language, some meaning on their own. Still presenting morphemes or single phones or vocalizations in isolation often convey no clear intention or meaning. Even for words presented in isolation, the intended meaning might vary a lot and the intention of the speaker may not become clear to the listener especially when words are realized with substantial reduction. Nevertheless, many words, even if spoken in isolation, do carry a relatively distinct meaning. Note, that we will focus in this thesis on spliced-out spoken word tokens that are extracted from conversational speech from non-professional speakers. Presenting these word tokens to human listeners has a high chance that the listener cannot understand or transcribe the word played to them. The word error rate can be as high as 50% to 80% for human listeners[4].

1.1.2 Meaning is start and end of speech production

Language and speech are a means to an end. It is a cultural development of the human race to efficiently communicate with mostly meaningful utterances. These utterances transfer observations, emotions, or facts about our surroundings, perceived world, or our inner feelings to another human being. Furthermore, speech can be recorded, stored, and played back later. This is very common for recordings of music songs, which usually have a high emotional information density. This view on speech and language emphasizes that there is a speaker (or sender), who encodes some meaning into the speech signal, and a listener (or receiver), who decodes the speech signal and constructs her own meaning from the signal.[83,13,45,44]

The focus on the meaning of the synthesized word introduces a many-to-one mapping. Different articulations and even different words, like synonyms are mapped to a similar semantic meaning. Nevertheless, the meaning constitutes an evaluation on how intelligible and useful the synthesized speech signal is. The evaluation on the meaning is in contrast to evaluations that work on the audio signal like acoustical measures or on the articulation of the articulators. Making the meaning important favors the synthesis of intelligible and meaningful speech and helps reducing the mimicry of irrelevant noise sounds in a target speech recording. In the end, the speech production model presented

here starts from a target meaning and ensures that this target meaning is encoded in the speech signal by listening and deriving the meaning from her own speech. In this sense meaning is the start and the end of the speech production process.

In PAULE, this is implemented by allowing error to modify the articulatory trajectories such that the resulting speech better expresses the intended meaning. As the meaning representation lexical fastText Library for efficient text classification and representation learning (fastText)[40] word embedding vectors are used. They are called semantic lexical word embedding vectors (semantic vectors) and are pre-trained for the German language from large text corpora in a bag of word fashion to place similar word meanings into similar regions of a 300-dimensional vector space, whereas different meanings are placed at large distances from each other in this space.

Using a 300-dimensional vector embedding is different from the multi-hot multi-label problem of language described in[90], where the multi-hot encoding uses a very high dimensional *sparse* orthogonal and binary outcome space of around 50,000 dimensions, the vector embeddings encode meaning into a high dimensional *dense* outcome space. Therefore, the embedding vector space is considerably smaller compared to the multi-hot encoding. In the end, the choice of the meaning representation is a design decision that has to be done for a specific implementation but does not affect the overall rationale behind PAULE.

1.1.3 Speech production is goal directed

Speech production is not only tied to meaning and inherently incorporates the transfer of meaning and, therefore, can be assumed to be signal, it furthermore tries to achieve the goal of conveying a specific (or broad) intention.[83,13,45] [2] After this intention is achieved, the next goal can be selected by the speaker. But if the goal of conveying the intended meaning is not sufficiently achieved, another utterance needs to be used to clarify the meaning. After all the speaker wants to be understood.

By focusing on the goal-directedness in the speaking process, the focus of the speech production process shifts from finding "the right articulatory movement pattern for a specific target meaning" to "are the articulatory movement successful in transferring the intended meaning". The articulatory movement will be defined by the control parameter trajectories (cp-trajectories), which control the articulatory speech synthesizer VTL (details follow in Chapter 2). What at first glance only appears to be a reordering of the same statement, changes the task from finding one set of (global) cp-trajectories for the target meaning, which will be the same in every context, to finding one set of (situational) cp-trajectories out of many possible ones that transfers the meaning to the specific listener.

Furthermore, rephrasing the task of speech production into a goal-directed problem allows for defining an (additional) error term in the goal space, which in PAULE is the word meaning. This goal space is with its 300 dimensions still large, but considerably

[2] Seeing the word as a signal is only a simplifying view on the matter and most likely cannot fully account for effects that speech has on human behavior or on how speech is used in inner speech.

smaller than the variably sized articulator space of typically 30 channels by around 200 time-steps for 0.5 seconds of speech, which leads to around 5,000 dimensions. Furthermore, the goal space has a better error landscape (and better gradients) to correct for hitting the right target.

Note that this principle of goal directed speech production does not require a specific goal representation and can be fulfilled with different implementations (cf. [68,20]). The goal space can be an acoustic, a semantic, or a somatosensory goal space to name three obvious choices for articulatory speech production [110,77,24]. Nevertheless, the decision in which goal space is optimized is crucial for the performance and the general abilities of the model. Optimizing in low dimensional acoustic goal space will result in the discovery of the vowel space [72]. In semantics, optimizing on a decontextualized fastText semantic embedding space is more robust and faster to train but should produce less nuanced speech compared to a multi-modal context dependent semantic embedding space.

1.1.4 Speech production is adaptive and error driven

Besides being goal-directed PAULE is designed around the principle of error-driven learning and error-driven behavior. Speech production in PAULE always and fundamentally relies (conceptually) on the back channel of the listener and the discrepancy of the meaning and acoustic quality of the word token produced, i. e. in the comparison of the finally synthesized word and its target meaning and target acoustics. This is called the outer loop of PAULE as it is the loop of planning a set of cp-trajectories, executing it, listening to the result, and using discrepancy between the resulting impression and the intended one as a feedback or error channel.

In addition, similar to the outer loop an internal error-driven error channel is used, which allows PAULE to adapt quickly to systematic distortions to the articulators and ensures smooth cp-trajectories between novel sequences of sounds or words. The internal loop uses a prediction or anticipation of the acoustics and meaning that PAULE plans to produce. This prediction is based on the experience with past articulations and the currently planned cp-trajectories. From this an error to the target acoustics and meaning can be derived and used to further correct the cp-trajectories without articulating or executing a single step through the speech simulator, hence the name internal loop (cf. [21]).

One advantage of the internal loop is also, besides its conceptual beauty, that it executes ten times to a hundred times faster compared to the articulatory synthesis with the speech synthesizer. This allows to iteratively go through the internal loop substantially more often. In each round, the whole word is imagined by PAULE, i. e. the predictive time horizon into the future is typically between half a second to one second of speech. The internal loop is conceptually similar to an "inner voice" during silent reading or while thinking through problems verbally. Furthermore, an indirect silent acoustic route boosts performance in visual comprehension [7].

For the correcting error signal, not only the semantic and acoustic prediction error of the internal loop is used, but additionally the effort required for the articulatory

movements is minimized. This results in a model that realizes the semantic and acoustic target in mind as closely as possible with the least effortful articulatory movements. Namely, the third time derivative, the jerk, is minimized to enforce periods of constant acceleration, which corresponds to phases of constant force (force = mass × acceleration; $F = ma$)[33,60,14] and the first derivative, the velocity, is minimized, which leads to stationary or constant cp-trajectories in time intervals where no change is required by the acoustic or semantic error.

Building a speech production system around the core idea of having an adaptive error-driven mechanism that drives the control of the articulators allows to naturally account for situational changes, like having to bite on a pen while speaking (or on a bite block in an experiment[35]), having a fixated jaw through the strip of a helmet while cycling or simply being drunk and don't feeling the tongue anymore. Furthermore, this allows adapting to silent or loud surroundings. To which extent the PAULE model achieves this has not yet been investigated, as a guiding principle the adaptive error-driven mechanism influenced the design process of and is implemented in PAULE.

1.1.5 Speech production is learned behavior

An error-driven and adaptive control model for an articulatory speech synthesizer can in principle be designed to find a globally optimal solution. In contrast, PAULE does not seek for the best planned cp-trajectories. Instead of finding the global optimum, a subjective optimum, given the experience of the instance of PAULE with spoken words and with producing words defines the error landscape and the gradients that lead to improved cp-trajectories.

Through this thesis, it is assumed that speech production is a learned process that at least involves the learned mapping between acoustic impressions of word tokens to meanings and the learned mapping between the own vocal tract movement patterns to the resulting acoustics.[83,13,45] In the light of the high dimensionality of the parameter space of the human speech system, it is not surprising that the production of a given word can be achieved by multiple, dissimilar, cp-trajectories. Furthermore, the learning history or the known relations between the articulatory movements and the resulting sounds introduce a bias on how to articulate newly learned words or words in a new linguistic environment.

Viewing spoken language and speech as a learned skill enabling human communication naturally results in the prediction of systematic dialects for second language acquisition, i. e. native German speakers have a different English accent compared to Japanese speakers and their English accent. The different accent results from systematic differences in the produced sound quality, manner of articulation, but also by over-using uncommon speaking variants or the higher probability of selecting uncommon words from the meaning space. These effect should be shown by any learning model of human speech production and are not original to PAULE. To which extend PAULE shows these effects is still untested.

This subjective optimization is an important and desired aspect of a model of human

speech production that tries to account for individual differences, but is not necessarily so desired if the technical task of solving the kinematic inverse problem for a specified target acoustics and target semantics is focused on. Here, global gradients that account for the full flexibility of the human speech system and give objectively optimal cp-trajectories would be better suited, but then this guiding principle is violated.

1.1.6 Articulation is influenced by experience

In addition to language experience and learning that systematically influences, e. g. second language acquisition, experience with the articulatory system itself and learning which combinations of articulation result in which sound patterns, are important to coordinate the different articulators in the production of different words or the same word in different contexts. This training effect in articulation is different, but not completely separate, to the influences of language use[108,106]. The training affects are tied closer to the movement and speech sound repertoire used[109], whereas the language use defines the meaning space and which acoustics are perceived to differentiate meanings, e. g. in minimal pairs.

During language learning, a model of speech production should be able to follow the slow but persistent changes in the vocal tract geometry from childhood to adulthood as well as account for the fact that most humans initially underarticulate words and over time, with continued practice, speech production comes ever closer to the norms of the learner's speech community until it is possible to even hyper-articulate words in late childhood and adulthood.

In PAULE the influence of articulatory experience is, firstly, reflected in the initialization or first guess of the cp-trajectories, which is conducted by a model that maps either from a word's target acoustics or a word's target semantics to initial cp-trajectories. Secondly, the correctness of the predictive forward model and its error gradients is better, whenever more articulatory experience is accumulated. Both of these effects should lead to higher coarticulation in high-frequency words compared to low-frequency words.

1.1.7 Sublexical units like the phoneme are not necessary (no phonemes)

Language and speech technology research has a long tradition in identifying, transcribing, counting, and categorizing spoken language into atomic, symbolic, or quasi-symbolic smallest sound units, so-called phonemes[56,111,101]. Phonemes were introduced in the late 19th century by Polish and Russian linguists Mikołaj Kruszewski, Lev Shcherba, and Jan Baudouin de Courtenay. Each phoneme (abstract speech unit) then can be realized as a phone (concrete speech unit). The transcription of spoken language into sequences of phones and phonemes and the counting of different phonemes used in a language allows for nice and transparent comparisons between languages and gives a measure of how reduced or articulated a specific chunk of speech is. Furthermore, differences in dialects can be quantitatively measured and advice can be given to people who need to train or relearn their speaking abilities. A phonemic transcription is a partially subjective endeavor and is influenced by the expectation of the transcriber, who

will introduce segments that are not realized and therefore not grounded in the speech signal[52]. A transcriber needs substantial training and high inter-rater reliabilities can only be achieved by agreeing on guidelines between transcribers. These guidelines are no general guidelines but have to be newly created for each data set. As well known from phonetics, the aligned data, being a sequence of phonemes alongside their respective durations, loses a large amount of information compared to the rich acoustic speech recording it is describing. Part of the lost information can be added by annotating for stress, pitch, or emotion, which is neither commonly done nor used in this thesis.

For a long time, phoneticians truly believed, and many still do (certainly phonologists) that the wild variation in actual speech can only be tamed by piping the signal through a phone-based representation, a recent example is the Shortlist B model of speech perception[67]. This is true for the task of text-to-speech systems as well, which involves finding the right sequence of phonemes and then applying different rules on how to recover a naturally sounding speech signal with the right intonation and a coherent speaker identity. A first step usually involves syllabification and finding chunks consisting of several phonemes that belong together. In the field of articulatory synthesis, in which the articulators of the human speech system are modeled besides the resulting speech signal, syllables or the smaller demi-syllables constitute an alphabet of reappearing sound units and reused movement commands (see the section below). Even if these approaches can account for coarticulation patterns, the underlying belief is that humans store a small number of sound units and then use combinatorial rules to either extract a word meaning during listening or to produce a word while speaking. These approaches have inherent problems to account for systematic differences for the same phoneme sequences in words with different meaning, the so-called homophones[37,61].

PAULE takes a radical approach and gets rid of all symbolic units. Therefore, PAULE does not contain any phonemes, syllables, or demi-syllables. Instead, it assumes that words are the smallest reasonable unit to investigate in the context of speech production. For languages, like Finnish or Chinese where the exact definition of a word is under debate, speech chunks of a duration between 0.5 to 2.0 seconds can be taken as long as a semantic vector embedding can be attached to it. Strictly speaking, PAULE is not relying on the word, but more general on chunks of speech with a duration starting from 0.5 seconds to 2.0 seconds to which a semantic word embedding vector can be assigned to. As long as around 20 hours of training data are given in this chunked-up form, PAULE should work to control the VTL for this language.

Phonemes might exist during the processing of speech by humans and might be a useful summary especially in analyzing speech data as it allows to easily search for similar sounds. Furthermore, many languages have an orthographic writing system and speakers of those languages learned how to read aloud or sound-out words from letters. Still, PAULE shows that the wide variety and flexibility of the spoken language, with usually more than 150,000 word types per language can be modeled without any phone representation.[3] Implementing PAULE without any phone representation gets supported

[3]PAULE is trained on distinguishing 4,311 different word types with a few high-frequency words and many low-frequency words in the German language. The data structures used in this implementation

by the successes of deep end-to-end models like Tacotron[117] and Wavenet[114] in recent years. These end-to-end models can produce high quality synthesis without any explicit phone representation. PAULE differs from these speech synthesis pipelines in the sense that it is a speech production framework that controls a simulation of the human vocal tract and, therefore, gives insights into how language might be executed and learned by humans.

The step to remove any phone representation and their combinatoric potential is rather drastic, even if good arguments can be made why phones and phonemes are a questionable level of representation to fully describe speech[75]. Therefore, we discuss in Chapter 7 what possibilities exist to add phoneme-like chunk representations back and how it might be possible to combine them in a productive and combinatorial fashion.

1.1.8 Gestural scores or movement targets are not necessary (no gestures)

In addition to having no phonemic, symbolic acoustic representation, PAULE will not have any symbolic gestural or movement chunks. Gestural scores and articulatory targets are less symbolic than phonemes, but still, they define the movement trajectories of the articulators, like the tongue, at critical points in time. Often, these descriptions of the human speech movements are defined for the full phoneme repertoire. Additionally, special gestural scores can be defined at the level of syllables or demi-syllables and are used when the (demi-)syllable occurs in a word.

Gestural scores and articulatory target positions for the articulators are a helpful tool to understand what the human speech system has to achieve to produce a specific set of sounds. These descriptions make it easy to pinpoint parts of the tongue and jaw movements that are necessary for a sound and those aspects or degrees of freedom of the human vocal tract that have a lot of flexibility even when a specific sound has to be produced. This flexibility can be used to model coarticulations.

As gestural scores and articulatory targets connect the articulatory movement of the human vocal tract to sound patterns, they have difficulties accounting for highly reduced forms (cf.[50,53]) that are common in natural speech and, similarly, to account for the high variabilities of the same word type, especially in the case of high-frequency words. As gestural score sequences are often derived from phoneme sequences, one problem is to model the presence of residual movements that are not acoustically visible. Assume a phoneme is acoustically dropped in the resulting reduced speech and the gesture of the tongue is still doing some movement in the direction of this dropped phoneme. How can this be resolved? One option to recover this residual movement is to add the phoneme into the phoneme sequence, even if it cannot be heard and cannot be detected in the acoustic signal, and to implement a smoothing algorithm between gestures that produces the residual movement. But other instances of the same word

are known to scale to all German words in different tasks. Therefore, even if more testing is needed for PAULE, we are confident that the model scales to the full word repertoire. Chapter 6 shows the evaluations done so far with and on PAULE.

might not show this residual movement, therefore, adding the phoneme in there, might not be the right choice. By deriving articulatory movements directly from the word meaning, this problem is not encountered.

Another practical reason not to use any quasi-symbolic gestural scores or articulatory targets is that it is not necessary to define all these symbols, which is tedious work and needs to be done on a speaker-by-speaker basis and for each language. Furthermore, it is not necessary to worry about reasonable mixing or blending methods between those gestural scores or targets. By not defining these targets, our model needs to be capable of handling the full flexibility of the human vocal tract. This is a challenge due to its high flexibility and, therefore, dimensionality and the non-linear nature of the human speech system. After solving the problem, PAULE allows to naturally produce highly variable and highly reduced movement trajectories for given word tokens.

However, at the moment the guiding principle of not having any gestures is achieved by using training data, which is created with a segment-based approach. The segment-based approach builds upon the tedious work of creating gestural scores for each phoneme in the German language. Therefore, the training data uses symbolic units in acoustics and in the motor space and appropriate, but not perfect, blending methods to mix the different gestural scores together over time. In the future, a goal-directed babbling approach can make the need for this training data obsolete. Even while being trained on the segment-based data, PAULE produces motor trajectories that show a different time dynamics compared to the segment-based ones and therefore moves away from the segment-based synthesis approach.

Another advantage of not defining any symbolic or quasi-symbolic units, like motor primitives on the movement side, is the ability of the model to start with unclear and under-articulated speech, which is hardly understandable, and then to gradually improve its overall articulation so that eventually it will be able to produce a word with a careful pronunciation, or even with a hyper-articulated pronunciation. The downside of not having symbolic units is that it might be (slightly) more difficult to generalize to new words, even if this is mitigated by learning a general mapping between sounds and articulation.

As we focus on some fundamental principles that we try to combine into a coherent model of single-word speech production, we make some more simplifications with respect to the motor domain: The human vocal tract model we use is a geometrical one, which define the positions of all articulators and all glottis parameters at each point in time. For this, the model requires 30 parameters for each point in time. We call these 30 parameters cps and the time-series of the cps the control parameter trajectories (cp-trajectories). A single time-step in this model is roughly equivalent to 2.5 milliseconds, which correspond to 110 samples taken from a 44,100 Hz audio signal. The geometrical model in contrast to a biomechanical model does not model the muscle contractions that are needed for the articulatory movements. The cp-trajectories define the shape and the position. Therefore, the geometric model is capable of articulating movements that are impossible to be performed by a real human being. Details on the vocal tract simulator we use are given in Chapter 2. Another simplification is that we treat all 30 cps of the

vocal tract simulator mathematically the same way. Therefore, the tongue position and the jaw position are mathematically treated equally, which might be reasonable, but they are also treated in the same way as the subglottal pressure, the velum opening, and the F_0-frequency of the vocal folds, to name a few of the 30 parameters.

1.1.9 The model should be as simple as possible, while still being able to model some main features of human behavior

The last two guiding principles have a more technical or methodological nature. PAULE should be as simple as possible, while still being able to model some essential aspects of the human speech production process. It is intended to be on a relatively high level of abstraction and, therefore, only approximates many of the fine details of speaking.

Being simple and simplifying should not result in precise predictions that contradict any known experimental findings. Furthermore, even if simple and simplifying the model should still be falsifiable. To the best of my knowledge, PAULE is not falsified yet, even if (of course) some of the findings in speech production might not be easy to be replicated with PAULE. PAULE as a whole should give a good account of how speech production could work on the behavioral side without considering muscle commands to the muscle systems in the human vocal tract, nor by accounting for the fine detail of the human ear and hearing system. Nevertheless, PAULE finds concrete cp-trajectories for a given target semantics and/or target acoustics, which allows the speech synthesizer to actually synthesize intelligible speech. In that sense, PAULE is a fully functioning and complete implementation of speech production.

The guiding principle of simplicity helps not only in creating the PAULE framework but additionally in selecting parts for the whole framework. The choice falls on parts that are substantially less complex, even if these parts might show slightly lower accuracies compared to complex parts. In the end, PAULE is not about the simple parts but about how they interact with each other and how they allow for an adaptive error flow and an internal and an outer loop to plan cp-trajectories. Having a simple and understandable model that is complete in the sense that it implements all required steps, hopefully, leads to a better understanding of the human speech process and how speech is different to written language.

1.1.10 Using time-series to time-series and time-series to fixed-vector models

The last guiding principle can also be seen as a first corollary of the no phoneme and no gesture principles. The whole speech production framework presented here is based on time-series to time-series models that are capable of modeling long-ranging relationships in high-dimensional (multiple channel) time-series data. Furthermore, models are used that collapse a time-series on a fixed-vector representation and models that unroll a fixed-vector onto a time series of requested duration.

This guiding principle avoids the necessity to find and model segments or syllable boundaries and there is no need for time warping or other methods from the dynamic

programming paradigm to match symbolic sequences on other symbolic sequences. Furthermore, this principle allows to model the dynamics in articulation and does not only explore the static configurations of the human speech system. Exploring the static configurations simplifies the problem substantially and mainly leads to the recovery of the vowel space [65,72,73].

In essence, the models in PAULE learn functions that establish mappings between acoustic and articulatory multivariate time-series data, and between static high-dimensional semantic representations and articulatory multivariate time-series data. The time-steps in the time-series data is selected to slightly oversample the underlying continuous trajectories. Therefore, no abrupt changes or jumps from one time-step to the next one are present and the time-series data is smooth and highly autocorrelated.

1.2 Human speech system

As PAULE is about *articulatory* speech synthesis, which approximates and models the human speech system, a short introduction on how the human speech system works is presented here.

Figure 1.1 shows a mid-sagittal slice of the human speech system. A description of a simplified version of how the speech signal is produced in humans follows:

Human speech production begins with the lungs, which create, through gravity or muscle contraction, a subglottal pressure that is higher than the supraglottal or ambient pressure of the outside air. The pressure difference below and above the glottis leads to vibration of the vocal folds. Depending on the pressure difference, the stiffness, and the mass of the vocal folds, (quasi-)periodic flaps are produced with a fundamental frequency F_0. The periodic flapping leads to periodic density differences in the air. For the speech or sound production, the supraglottal density changes are the important ones, as they travel through the oral cavity and potentially through the nasal cavity, where they are modulated and finally are emitted through the mouth or nose as a wave signal.

Some of the sound is transmitted through the skin while speaking. Therefore it is possible to create communicative sounds even while mouth and nose are closed, even if this is only possible for a short period of time as the pressure difference between subglottal and supraglottal pressure can only be achieved for a short time period.

The modulation harmonics of the fundamental frequency of the sound waves is given through the shape and modulation of the shape of the oral cavity. The harmonics or resonance frequencies are called formants in the speech signal. Formant frequencies are determined by the shape of the oral cavity. Humans modulate the oral cavity mainly through jaw and tongue movements and less through the lips, raising of the sound box, and by closing or opening the velum.

A useful view of the physics of sound production is to separate the process into two components, the noise sources of the signal and the filters that modulate the noise sources. The sources modulated by the filters result in the final waveform that the listener hears. In this source-filter-model, the changes in sources and the changes in the filter are independent of each other. For many sounds humans can articulate, this

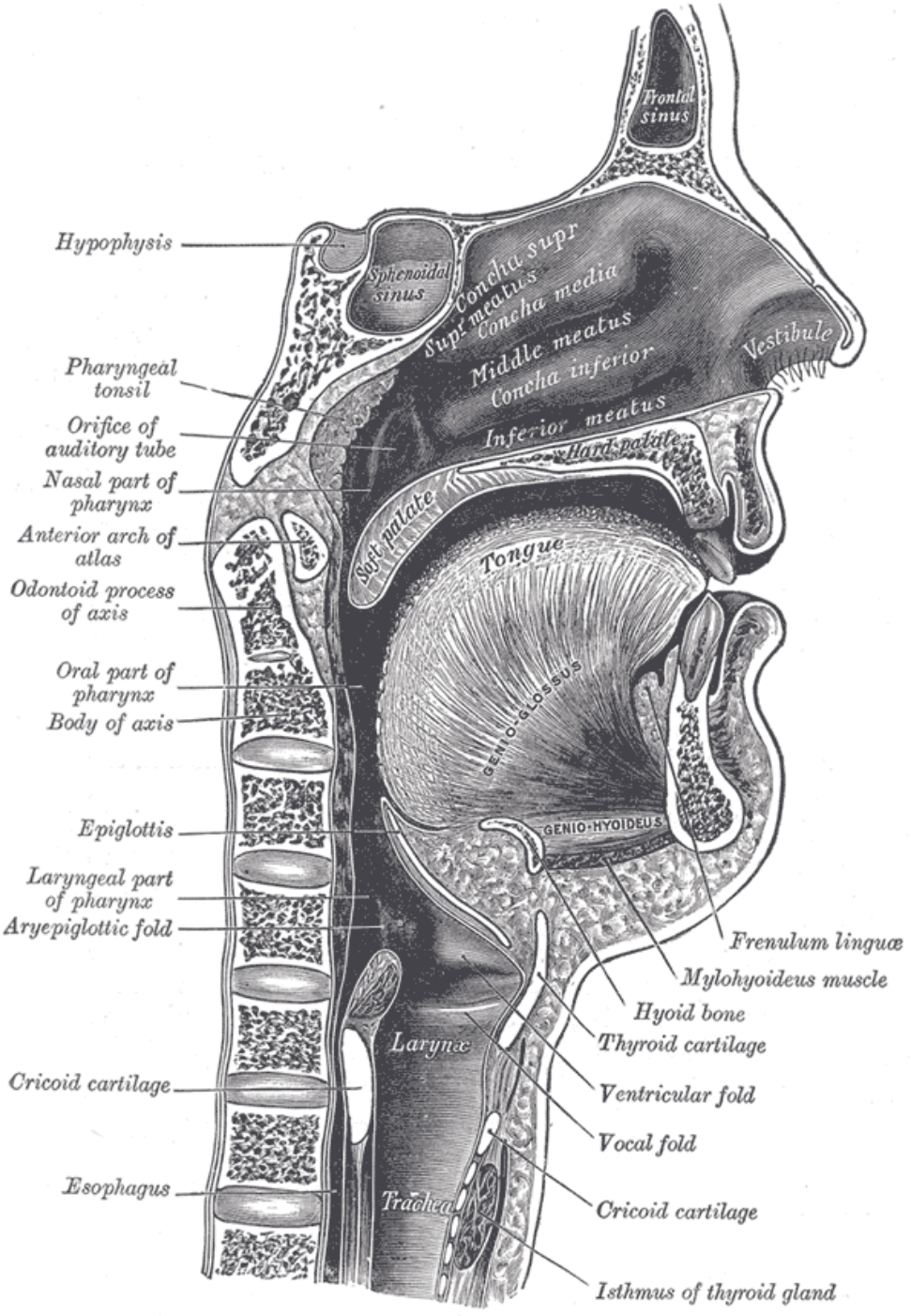

Figure 1.1: Mid-sagittal slice through the human speech system. [(c)]

[(c)] public domain; via `https://de.wikipedia.org/wiki/Datei:Sagittalmouth.png`

is true as the main source are the vocal fold vibrations and the filter is mainly defined by the position of the tongue, lips, and jaw. At small constrictions or closures in the oral cavity, which result in fricatives like the [ʃ] sound in the word /**sh**ould/ or the [s] sound in /mi**ss**ing/ or clicks, a sound source needs to be added to the F_0 sound source of the flapping vocal folds. Therefore, for fricatives and clicks the independence between source and filter does not hold in practice. Still, even these processes can be modeled with the source-filter-model by adding the necessary sources to the model manually, whenever the filter gets into a specific regime.

A nice aspect of the source-filter model is that it gives distinct meaningful properties to the sources and filters. In general, sources add sound energy, fundamental frequency (also called pitch) and sound quality like whispering and aspiration, whereas the filter imprints on the source signal formant modulations and removes different amounts of energy through dampening, but cannot add any energy to the final signal.

Formulating the problem of articulatory speech synthesis to a source and filter model already simplifies the problem, as there exist many different vocal tract configurations that lead to very similar filter characteristics and in certain situations a small change in the vocal tract configuration results in a huge difference in the filter characteristic. Therefore, it is tricky to derive articulator positions from a source-filter configuration.

While we focus here on the aspects of speech production and how the lips, mouth, nose, tongue, and the rest of the articulators are used in this process, it should not be forgotten that all these parts have to serve other purposes as well. The main overlap for most of the parts used during speaking are with breathing, to regulate the oxygen level in the blood stream, and eating, to have enough intake of energy and spare parts to repair and build up the human body.

In this thesis, the VocalTractLab (VTL) articulatory speech synthesizer is used. The central problem for which PAULE is one possible solution (out of many) is how to control this particular articulatory speech synthesizer and how to solve the many-to-one problem: How many different possible cp-trajectories result in similar or the same acoustics. The VTL is presented in detail in Chapter 2.

1.3 The human ear

The human ear is not the focus of this thesis. Nonetheless, approximating the human hearing abilities and mapping the sound pressure changes or waveform of the speech signal into a perceptual measure, the so-called acoustic representation, is one critical source of error and feedback that helps ensure the quality of the synthesis in PAULE.

PAULE distinguishes clearly between physical, objective measures that can be reliably and precisely measured and psychological measures that are subjective, i. e. undergo some individual differences, and are shaped by the human perceptual system. As an example, the pressure differences of the speech waveform are a physical measure, as are the frequency components in the speech signal and its energy or magnitude. In contrast, pitch and loudness are psychological measures, which are approximated here by Mel-frequency bands and log-magnitude values. This is described in detail in Section

3.2.

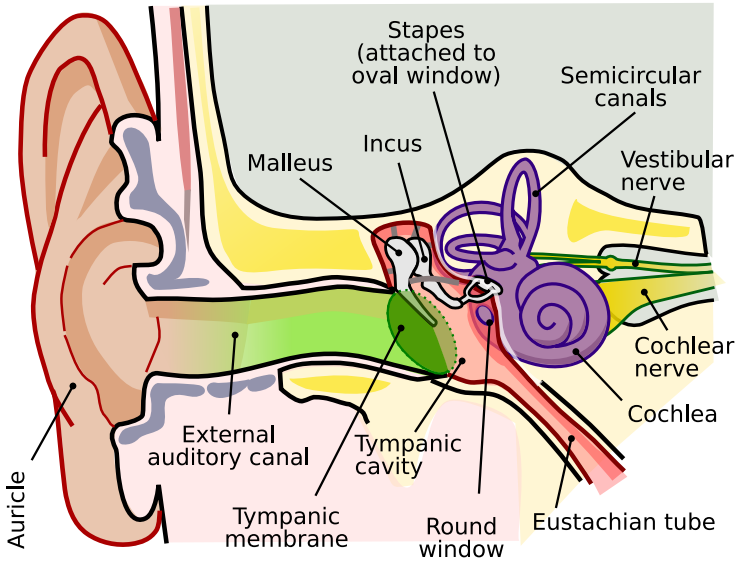

Figure 1.2: The human ear. [c]

[c] (cc)-by Lars Chittka; Axel Brockmann via
https://upload.wikimedia.org/wikipedia/commons/d/d2/Anatomy_of_the_Human_Ear.svg

The human ear (see Figure 1.2) and the listening process work roughly in the following way:
The sound waves as pressure differences propagate with the speed of sound through the air until they reach the head and outside part of the ear, the auricle, of the listener, where the sound waves are partly reflected and amplified. The original sound waves together with the reflections travel down the external auditory canal, where at the end, the pressure changes in the sound wave bring the tympanic membrane into a swinging motion. This swinging motion then is amplified by the middle ear and is finally transferred to a liquid in the inner ear. The inner ear then translates the mechanical motion through hair cells and a tube-like geometry into frequency decomposed electrical nerve signals. As in the middle ear, in this process, the signal undergoes some enhancement and amplification.

All the nerves that code the frequency-decomposed pressure signal are collected and then routed to the brain to connect and share information with the second ear in a very early stage to locate and decompose sound sources with two-ear listening. From here, the signal is routed to different parts of the brain that are specialized in analyzing and integrating the signal with other senses. For the purpose of this thesis, we will only focus on and approximate the function of the ear and heavily simplify the higher level processing of the brain with a single mapping that maps the acoustics to a semantic representation.

The healthy human ear can perceive sound waves in the frequency range of 10 Hz and 20,000 Hz. The sensitivity of the human ear gets smaller and smaller in terms of absolute frequency steps measured in Hz for higher frequencies. A 20 Hz tone can be clearly distinguished from a 30 Hz tone, but a 15,000 Hz tone is not or only barely distinguishable form a 15,010 Hz tone. The decrease in sensitivity follows roughly a linear decrease up to 1,000 Hz followed by a logarithmic decrease[103,113,2].

1.4 Goal of the thesis

In summary, the goal of the thesis is to develop an articulatory speech production framework, which I term PAULE, that produces word utterances without relying on symbolic motor (gestural scores; motor primitives) or sound units (phones). The start of the speech production should be a target semantic embedding or target acoustics. The resulting articulatory movement trajectories should fulfill some physical localisation and constant-force constraints and should be within the range of what a human is capable of doing with their vocal tract. The main challenge for model building is to bring the high dimensionality of the vocal tract and its non-linearities under control and solve the many-to-one mapping of how many different possible cp-trajectories result in similar or the same acoustics.

1.5 Thesis overview

This thesis is divided into seven chapters. After this introduction (Chapter 1), articulatory speech synthesis in general and the VTL speech synthesizer specifically are described in Chapter 2. In Chapter 3, the different data structures and data transformations on the semantic, articulatory, and acoustic domain are introduced and defined. The data structures are filled with actual data in Chapter 4, where different data sets and corpora are introduced. The main chapter is Chapter 5, where the PAULE model is introduced and described. This is followed up with results (Chapter 6) and a discussion (Chapter 7). The discussion includes a conclusion at the end.

2 Articulatory speech synthesis

(...)
Du hast nie gelernt dich zu artikulieren
Und deine Eltern hatten niemals für dich Zeit
Oh oh oh, Arschloch
(...)

Schrei nach Liebe, die Ärzte, 1993

As a first building block towards introducing the Predictive Articulatory speech synthesis Utilizing Lexical Embeddings (PAULE) framework, it is important to understand what an articulatory speech synthesizer is, which one we use and with which approximations this speech synthesizer comes. The articulatory speech synthesizer is so important because PAULE can be seen as a cognitively motivated control model for an articulatory speech synthesizer. Therefore, the main goal of the PAULE framework is to find suitable inputs to an articulatory speech synthesizer that results in the production of speech, which has a desired meaning and a desired acoustic quality.

2.1 Short history of speech synthesis

Humans have a long tradition of building and controlling *speaking machines*. For a long time, only mechanical devices were possible. Mostly, these speaking machines could be separated into two components. One component would generate a period tone or sound and corresponds to the human voice box with the glottis. Another tube-like geometry modulated this sound to produce speech-like sound patterns, which correspond to the oral and nasal cavities in humans.

Building machines like the Sprechmaschine by Kempeln (see Figure 2.1) allowed for an analysis by synthesis approach to speech, which tries to understand human speech production by resynthesizing the speech signal with machines or models. Already in the eighteenth centuary, this analysis by synthesis approach led to the insight that human natural speech is highly dependent and influenced by coarticulation patterns, i. e. each speech sound sounds systematically different depending on the speech sounds surrounding this sound. Furthermore, this results in the insight that it is not enough to build a sound or phone repertoire and concatenate these together. Besides the scientific nature of building these speaking machines, they were a fascinating attraction to people. For a longer abbreviated history of speaking machines see Story[104].

With the rise of electric circuitry and electrical sound speakers, electrical devices were constructed to control the electrical loudspeaker – without any articulation – to

Figure 2.1: A replica of a speaking machine designed and constructed by Wolfgang von Kempelen around 1790. [(c)]

[(c)] (cc)-by-sa Fabian Brackhane via
https://commons.wikimedia.org/wiki/File:KEMPELEN_Speaking_Machine_Replica_2017.png

synthesize a speech signal. Vocoders were devices that specialized in encoding a speech signal into an electrical analog signal and then back into a speech signal from the analog signal. They were specialized in the sense that they could not efficiently encode, e. g. music. This is in contrast to radio broadcast signals, which were relatively agnostic to the input in a given frequency interval. Vocoders found applications in the production process of music, in the military, and in science.

The rise of the digital era and the availability of digital-to-analog and analog-to-digital converters in the 441,000 Hz range fundamentally changed speech synthesis and speech encoding technology. Even if most telephone lines in the time of the compact disk (CD) still ran on an analog signal, the field of speech synthesis was dominated for a long time by models that used so-called statistical synthesis. As for the vocoders, articulation is not modeled in statistical speech synthesis, but a database of recordings of sounds and syllables from a given target voice is used to build up the speech of a target utterance. Concatenating all these sounds results in a very clunky and weird-sounding signal, which tended to have a lot of unwanted clicks in it. But after some smoothing, filtering, and other signal processing techniques, the speech synthesis sounds like the typical computer voice from the 1990s.

The statistical synthesis was followed up by direct wave approximations like Tacotron[117] and WaveNet[114]. Again these speech synthesis pipelines directly model the resulting speech signal and try to solve the text-to-speech problem without modeling any articulation. These models were trained in an end-to-end fashion and did not work with theoretical linguistic constructs such as phonemes or syllable, and yet they can produce very naturally sounding speech signals. These end-to-end direct speech wave approximation models, as well as hybrid models combining the last approaches with the statistical synthesis, are the models that are used in modern days text-to-speech synthesis systems, and can be used in daily life. Note, however, that these approaches do not give any insight into how humans use their speaking system to produce speech.

In order to study human articulation and the physics involved in the human vocal tract, articulatory speech synthesis systems have been developed. Until today, they do not sound as natural as statistical or hybrid systems. One reason is that most of them have a tube-like sound, which comes from a 1-dimensional tube approximation of the 3-dimensional human vocal tract. The big advantage of articulatory speech synthesis systems is that they model the articulatory movements and from these synthesize the acoustics. Therefore, they give direct insight into the speaking process. Examples for articulatory speech synthesis systems are the *Haskins Configurable Articulatory Synthesizer* (CASY)[48] with HLsyn[43], the Maeda synthesizer[62], and the TubeTalker[104] as well as the VocalTractLab (VTL)[15], which is used in this thesis. Models that model the articulatory control with some feedback loops are the *Task Dynamic model of inter-articulator speech coordination* (TADA) model[66], which uses CASY as a synthesizer backend, DIVA[110], which uses a variation of the Maeda synthesizer, and FACTS[69], which makes use of CASY as its synthesizer. Our PAULE model, therefore, is comparable to TADA, DIVA, and FACTS. Yet another approach was taken by the OPAL group of the ArtiSynth[29] project, which developed a high-dimensional biomechanical model of the human vocal tract, models the human vocal tract. However, this model is not coupled with an acoustic synthesizer.

Both, the CASY and the Maeda synthesizer use a 2-dimensional representation of the human vocal tract with only a few degrees of freedom and a 1-dimensional simulation of the acoustic wave. These 1-dimensional simulations simplify the problem drastically and keep computation times tractable. The 1-dimensional acoustic simulation uses a tube approximation. This means that the oral cavity between the voice box and mouth is approximated with a sequence of circular or elliptic tube segments of varying diameters. This varying diameter results in differently sized areas and the characterization of the complete sequence of tubes is called the area function. From this area function, either pressure differences are calculated in the time domain or resonances in the frequency domain, or a mixture of both. In the end, a wave audio signal is created.

To evaluate the quality of the 1-dimensional approximations of the 3-dimensional human vocal tract, 3D-printed tube-approximated vocal tract shapes can be excited and recorded in the laboratory and compared with the simulations at the computer[34]. Figure 2.2 shows the vocal tract tube approximation printed with hard plastic (PLA) and silicon. Both of these were measured and compared to the computational simulation by

Figure 2.2: 3D-printed vocal tract tubes for the vowel /a/ by [34]. Both tubes have the same shape but consist of two different materials, one is out of hard plastic (PLA) and the other one is out of silicon. The /a/ sound these tubes produce is measured and can be compared to 1-dimensional acoustic simulations. [(c)]

[(c)] (c) TU Dresden Peter Birkholz via `https://tu-dresden.de/ing/elektrotechnik/ias/stks/die-professur/news/messung-der-transferfunktion-von-rohrmodellen-des-vokaltraktes`

the group around Peter Birkholz at TU Dresden.

As both the Maeda synthesizer and the CASY synthesizer only use a 2-dimensional mid-sagittal shape with a relatively small number of free parameters it does not produce high quality speech and has to do manually allow for airflow along the sides of the oral cavity like in the sound [l]. Furthermore the small number of free parameters comes with less flexibility and therefore with less potential to achieve higher synthesis quality with a different control model. The TubeTalker requires the Matlab environment, and reimplementation in python proved prohibitively time-consuming. VTL has support for the German language, for which we had articulatory data, and was easy to interface from the Python programming language, hence the choice was made for VTL. In principle, it should be possible to exchange the VTL with any other articulatory speech synthesizer.

2.2 VocalTractLab (VTL)

In order to understand PAULE and to define the problem scope, it is important to understand the abilities and limitations of the articulatory speech synthesis system,

which is controlled by PAULE. For PAULE, we choose the VocalTractLab (VTL)[15] [1] synthesizer as it is the most complete, flexible, and feature-rich articulatory speech synthesizer currently available. Furthermore, it is able to synthesize the German language and is Free and Open Source Software (FOSS) under active development[38]. The German synthesis allowed us to subjectively and easily evaluate synthesis quality as German is my mother tongue. That the VTL is FOSS allowed us to adjust its code to automate the extraction of several parameters of interest. Some of the code contributions were merged upstream into the official version of the VTL.

As VTL is under active development, over the time of the thesis, VTL was used in different versions, namely 2.2 to 2.3, and in the end a modified version, 2.5.2quantling, which is the version of VTL that is distributed with the `paule` python package to allow for reproducible results. As VTL is Open Source Software, the source code to the 2.5.2quantling version of VTL can be found on GitHub[2] as well as the source code for the PAULE framework[3].

In PAULE, the articulatory speech synthesizer is treated as a black box. The inputs to the VTL are 30 real-valued input control parameters (cps) over time, so-called control parameter trajectories (cp-trajectories). The cp-trajectories define the cp every 110 samples of the resulting 44,100 Hz mono audio signal. The 44,100 Hz mono signal is the result of the geometric and acoustic modeling in the VTL and is the output of the speech synthesis. The 110 samples per 44,100 Hz results in a time-step of roughly 2.5 milliseconds. As the duration of the speech signal can have variable length, the number of time-steps is variable and usually between 50 and 400 time-steps for a single word in German.

VTL approximates and simulates the physical pressure changes of a 3-dimensional vocal tract (see Figure 2.3) in a quasi 1-dimensional acoustic simulation. that simulates the pressure differences in along the oral and nasal cavities. The shape of the oral cavity is determined by the 3-dimensional geometric model. VTL therefore approximates the physical properties of the human speech organ but does not try to emulate any muscle commands. The geometrical positions of the tongue and jaw, as well as the properties of the glottis and the subglottal pressure, are directly defined for each time-step.

In order to synthesize speech with VTL, a configuration or speaker file has to be loaded. Throughout the thesis the standard speaker of VTL was used. In VTL 2.5.2quantling, this is speaker *JD3.speaker*. The speaker file defines the fixed geometry of the vocal tract as well as gestural scores for all German phonemes. In this thesis and in PAULE, the gestural scores are used to generate some initial training data and to compare the synthesis created by PAULE to the segment-based approach but are not used anywhere else (see Section 4.1 for details).

The acoustic simulation in the time domain uses a tube approximation of the oral cavity. Along the midline of the cavity variably sized tube sections are inserted. Each tube section has an area associated with it. This approximation allows for a 1-dimensional simulation of the pressure differences of the longitudinal wave and its resonances in

[1] `https://www.vocaltractlab.de/`
[2] `https://github.com/quantling/VocalTractLabBackend-dev`
[3] `https://github.com/quantling/paule`

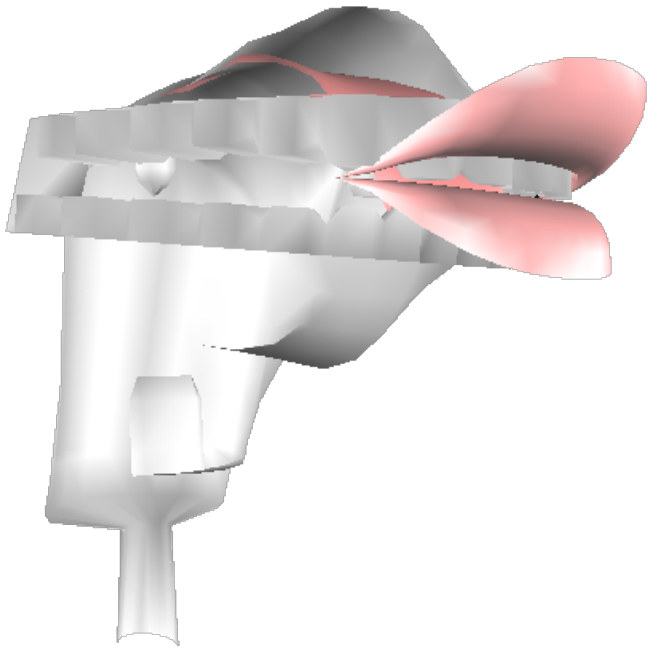

Figure 2.3: The VocalTractLab uses a 3-dimensional geometric model of the human vocal tract and synthesises the pressure differences that are emitted by mouth and nose with a quasi 1-dimensional acoustical model that simulates the pressure differences in along the oral and nasal cavities. The shape of the oral cavity is determined by the 3-dimensional geometric model.

the cavity. The oral cavity is not approximated by just a single tube, but by a tube that branches into side cavities directly behind the glottis and below the tongue tip. Additionally, the nasal cavities are connected to the tube by branching into the nasal cavity. The nasal cavity is modeled with several Helmholtz resonators.

Modeling speech production with a tube-like model has a long tradition. The advantage of the tube approximation is that a 3-dimensional wave and resonance modeling problem can be reformulated into a 1-dimensional one, which is substantially easier to compute. Reducing the problem space of the physical pressure simulation in this way only allows for simulation of the longitudinal wave along the tube middle line. This is a good approximation up to 8,000 Hz as wavelengths below 8,000 Hz do not fit into the cavity of the mouth transversally and are, therefore, nearly completely damped out. The completely damped out components of the signal do not need to be modeled in the first place therefore ignoring transversal waves is a good approximation below 8,000 Hz. For a frequency above 8,000 Hz, the wave length of transversal sound waves starts to fit into the diameter of the oral cavity and has an effect on the finally emitted sound and, therefore, the approximation of ignoring transversal waves degrades in this frequency region. However, most of the time, the degradation will not play an important role in speech however, as the important features of the speech signal is mostly encoded

between 300 Hz and 3,000 Hz. Telephone companies made use of this fact and encoded the transferred signal only in this narrow band. Still, even if speech is mostly intelligible the sound quality is changed and degraded by limiting the signal to the narrow telephone band. This is especially salient for fricative sounds in German like the [ʃ] in /**sch**ön/, the [z] in /**S**oße/, the [s] in /So**ß**e/, the [ç] in /i**ch**/, the [x] in /Tu**ch**/, and the [h] in /**h**eute/), All these sounds are difficult to distinguish if dicateted through the telephone line. Increasing the frequency band to the range of 20 Hz to 8,000 Hz seems to be wide enough to capture all of the main features of speech and many speech research downsample their data to a 16,000 Hz wave signal. With 16,000 Hz some features of fricatives are still removed and to keep all information that can be perceived by the human ear it is necessary to have a sampling rate of at least 44,100 Hz or 48,000 Hz.

As described in Section 1.3, which discusses the human ear, the audible frequency range goes up to 20,000 Hz for healthy humans. This seems to be in contrast with the observation that for, at least most western languages, the frequency range up to 3,000 Hz seems to be sufficient for communication and the range up to 8,000 Hz seems to capture most of the qualities of the speaker's voice. This can be further understood by the fact that the sensitivity in higher frequencies is lower in terms of absolute frequency steps and that it is difficult for humans to produce high-frequency sounds. For men the F_0 is lowest and usually lies in the range of 100 to 120 Hz, for women the F_0 is around 200 to 240 Hz and for children around 300 Hz. For intelligibility the F_0 plays a subordinate role, more important are the characteristics of the formants for vowels and the spectral properties in the frequencies in the range of 2,000 Hz to 4,000 Hz for consonants in non-tonal languages like German. Our ear can hear a lot higher frequencies than we can produce, which is no surprise as the wider range in listening allows us to detect and listen to sounds produced by small animals, like insects and birds, and gives us a better ability to distinguish different cracking and colliding sounds.[36]

Figure 2.4 shows the mid-sagittal slice of the oral cavity of the VTL as well as a depiction of ten of the 30 cp-trajectories and the resulting mono audio as a waveform. Figure 2.3 shows the 3-dimensional mesh model of the VTL vocal cavity. The nasal cavities and the lungs are not shown but are part of the synthesis process.

The VTL comes with a graphical user interface (GUI) to orchestrate gestural scores and to show different aspects of the human vocal tract as a teaching tool. This GUI is not used in this thesis.

As the PAULE framework is cognitively motivated, it accumulates experience on the cp-trajectories to acoustic representation mapping. This knowledge is later used to improve the synthesis and find cp-trajectories for a target semantics or a target acoustics for a given word type. In the following Chapter 3, a precise description of the input parameters of the VTL, the cp-trajectories, and the other data structures used in PAULE are given.

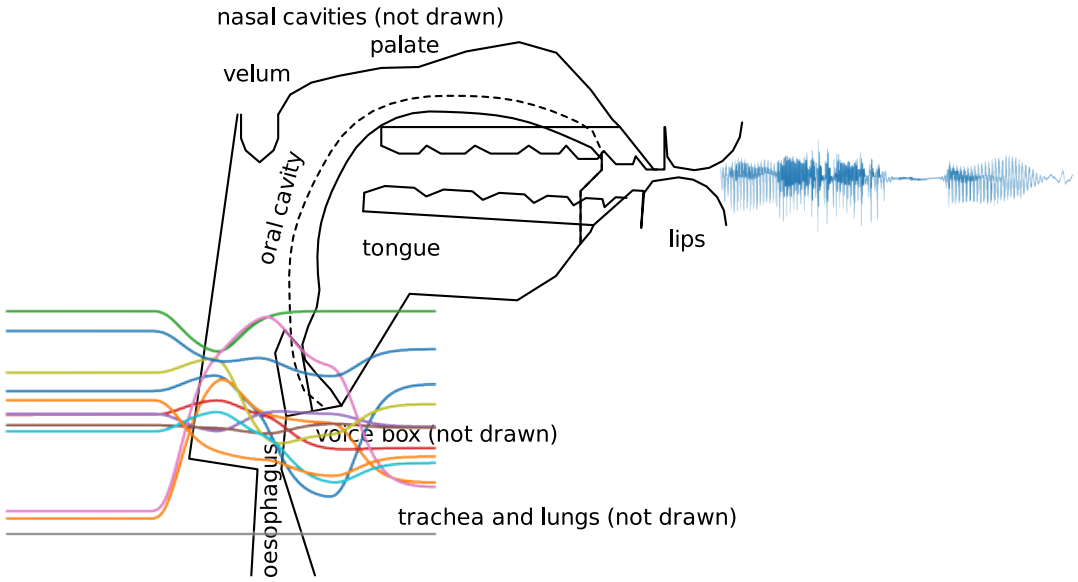

Figure 2.4: The VocalTractLab (VTL) receives control parameter trajectories (cp-trajectories) as inputs and outputs a mono audio signal at 44,100 Hz.

3 Data structures

> Smart data structures and dumb code works a lot better than the other way around.
>
> Eric S. Raymond

Before we start combining and building up the PAULE framework around the VTL speech synthesizer (cf. Chapter 2), the different data structures and the rationale behind them are introduced first. In order to understand the challenges and intricacies of the PAULE framework, it is important to understand the decisions that went into these data structures. This chapter is restricted to a discussion of the structure of the data and does not touch upon the data sets themselves – these will be introduced in Chapter 4. The data structure tells us, which information is *in principle* presented to the different models and mappings, whereas the data sets tell us, which information is *actually given*. The actually given information is always a subset of the information that could in principle be encoded into the data structure.

The current version of PAULE, the subject of this thesis, does not take visual information into account. It also does not receive somatosensory feedback. It cannot feel or taste the tongue or lips and does not know where the tongue is in contact with other articulators. Furthermore, it does not extract any meaning from the articulatory movement. The model only extracts meaning from the acoustics and cannot handle silient speech. Therefore, the model is a little bit like a blind person that has no feeling in its articulators and only uses the acoustic information as a source for the meaning of the uttered word. A topic for further research is to integrate visual and somatosensory information into the PAULE model.

The data structures are chosen to slightly oversample the underlying signal and/or to be of slightly higher resolution of what is commonly done in speech research. This decision should present the model with more information than what is needed and should foster the ability to let PAULE select relevant information, while still keeping computational costs reasonably low. Increasing the resolution of the data structures is expected to slow down computations in PAULE considerably without leading to a substantial increase in accuracy.

All models are trained and all results are calculated on the data structures presented below. These data structures are human-readable, theoretically motivated, and work well within the PAULE framework.

3.1 Control parameter (cp)-trajectories

The inputs to the VTL simulator over time, the so-called control parameter trajectories (cp-trajectories), are the central data structure in this thesis. The PAULE model is given the task to find optimal values for the cps, across three different tasks: mimicing (acoustic only; copy-synthesis), meaning-driven speaking (semantic only), and speaking driven by both mimicing and semantics (semantic-acoustic task). In the configuration used throughout this thesis, the cp-trajectories have 30 channels and are defined every 110 samples of the produced 44,100 Hz audio signal therefore roughly every 2.5 milliseconds ($\approx 110/44100Hz = 11/4410s \approx 0.002494s$).

As control parameters or channels have different purposes and define different aspects of the speech synthesis model, different units are attached to them and they operate on different scales. To make all parameters comparable to each other for each channel its central value ($(max + min)/2$) is subtracted and this centered value is divided by the theoretical range ($max - min$) of the channel. The resulting channels are all distributed around zero with a range of roughly -1 to +1 (Listing 3.1). The names and units of all 30 parameters can be found in Table 3.1 and in Table 3.2.

As this normalization step is a linear transformation with a known and theoretically grounded intercept and scaling factor, the inverse is easily computed and does not lose any information. Still, the normalization achieves that all channels are unitless and comparable in size to each other and therefore (linearly) combining different channels can be justified.

Listing 3.1: Normalizing the cp-trajectories

```
1 import numpy as np
2
3 from paule import util
4
5 # allocate 5 time-steps of "cps"
6 cps = np.ones((30, 5))
7 normalized_cp = util.normalize_cp(cps)
8 inv_norm_cps = util.inv_normalize_cp(normalized_cps)
9 assert np.allclose(cps, inv_norm_cps)
```

Although not all of the 30 control parameters work the same way, they receive identical treatment in our model. In this sense, our model is generic. Possible improvements by differently treating different cps channels is a topic left for future work.

During our modeling work, three different major versions of VTL were released. Each verion of VTL slightly changed the physical interpretation of the channels in the cp-trajectories. The PAULE model in its configuration presented here, was trained and tested on two major versions. Results were very similar, which makes us confident that PAULE is relatively agnostic about future changes to the VTL and might even transfer well to another articulatory speech synthesis system.

There are three distinct types of cp-trajectories used in PAULE. These are described in detail in the following subsections.

Table 3.1: First 19 parameters out of the 30 parameters (channels) of the cp-trajectories of the VTL are the vocal tract parameters.

Vocal Tract Parameters				
Abbr.	Name	Min.	Max.	Unit
HX	horizontal hyoid position	0.0	1.0	*cm*
HY	vertical hyoid position	-6.0	-3.5	*cm*
JX	horizontal jaw position	-0.5	0	*cm*
JA	jaw angle (deg.)	-7	0	*deg*
LP	lip protrusion	-1.0	1.0	*cm*
LD	lip distance	-2.0	4.0	*cm*
VS	velum shape	0	1	–
VO	velic opening	-0.1	1.0	–
TCX	tongue body horizontal (X) position	-3.0	4.0	*cm*
TCY	tongue body vertical (Y) position	-3.0	1.0	*cm*
TTX	tongue tip horizontal (X) position	1.5	5.5	*cm*
TTY	tongue tip vertical (Y) position	-3.0	2.5	*cm*
TBX	tongue blade horizontal (X) position	-3.0	4.0	*cm*
TBY	tongue body vertical (Y) position	-3.0	5.0	*cm*
TRX	tongue root horizontal (X) position	-4.0	2.0	*cm*
TRY	tongue root vertical (Y) position	-6.0	0.0	*cm*
TS1	tongue root side elevation	0.0	1.0	*cm*
TS2	tongue back and dorsum side elevation	0.0	1.0	*cm*
TS3	tongue tip and blade side elevation	-1.0	1.0	*cm*

Note: The first column gives the abbreviation (Abbr.), the second column the full name, the third the minimal value (Min.), and the fourth the maximal value (Max.) and the last column contains the unit.

Table 3.2: Last 11 parameters out of the 30 parameters (channels) of the cp-trajectories of the VTL are the glottis parameters.

Glottis Model Parameters				
Abbr.	Name	Min.	Max.	Unit
F0	fundamental frequency	40	600	*Hz*
PR	transglottal pressure	0	20000	*dPa*
XB	lower edge of vocal folds	-0.05	0.30	*mm*
XT	upper edge of vocal folds	-0.05	0.30	*mm*
CA	posterior glottal chink area	-0.25	0.25	mm^2
LAG	phase lag	0	3.1415	*deg*
RA	relative amplitude of oscillation	-1	1	–
DP	diplophonic double pulsing	0	1	–
PS	skewness of glottal area pulses	-0.5	0.5	–
FLUT	small quasi-random fluctuations added to F0	0	100	%
AS	aspiration strength	-40	0	*dB*

Note: The first column gives the abbreviation (Abbr.), the second column the full name, the third the minimal value (Min.), and the fourth the maximal value (Max.) and the last column contains the unit.

3.1.1 Initial cp-trajectories

The initial cp-trajectories are the ones which PAULE either derives or already has available at the beginning of the planning process. The initialization process is different for different tasks and, therefore, can be the result of different internal mappings. These different mappings are implemented within PAULE as different internal models (cf. Section 5.4). For example, initial cp-trajectories can be derived from a target acoustics with the direct inverse model or be can be generated from a meaning representation by the Wasserstein-GAN trained to sample cp-trajectories (cp-GAN). Furthermore, the end user is given the option to pass on initial cp-trajectories to PAULE.

The initial cp-trajectories are used and needed by PAULE to start the planning process and as PAULE locally optimizes the trajectory with some minimal effort constraints, different initial cp-trajectories are expected to result in different planned cp-trajectories even for the same target acoustics and target semantics.

3.1.2 Planned cp-trajectories

The planned cp-trajectories are the final result of the planning process by PAULE. Starting from the initial cp-trajectories, the planned cp-trajectories are optimized to minimize the Root Mean Squared Error (RMSE) between a predicted acoustic representation and semantic representation and their respective target acoustics and target semantics, while

keeping the effort in terms of velocity and jerk of the cp-trajectories small.

The planned cp-trajectories therefore depend on the target acoustics and the target semantics, as well as on the initial cp-trajectories and the number of iterations of the planning procedure. The planned cp-trajectories are only deterministic in the sense that, with the same models in PAULE, the same number of planning iterations, the same shuffling in the continued learning of the forward model, and the same targets and initial cp-trajectories the same planned cp-trajectories are expected.

Synthesizing speech by using the planned cp-trajectories as inputs for the VTL results in the produced signal, a 44,100 Hz mono audio wave form. From this produced signal the produced acoustics can be derived, which corresponds to the human perception of the produced signal. From the produced acoustics the produced semantics can be derived. This produced acoustics and produced semantics can be compared to the predicted acoustics and predicted semantics.

3.1.3 Segment-based cp-trajectories

The segment-based cp-trajectories play only an indirect role in PAULE. They come from a segment-based synthesis approach described in Sering et al.[94] and in Section 4.1. A corpus of segment-based cp-trajectories with their associated acoustic representations is used to train the internal models of PAULE. As they are derived from a phone segment sequence together with the corresponding durations, they show repetitive patterns in the articulatory movements. The blending method used to combine the different gestural scores of the phone segments allows for some coarticulation but does not show the variability that is present in the planned cp-trajectories that PAULE produces as optimal[95].

The segment-based cp-trajectories are mentioned here, as they comprise a reference to which the planned and initial cp-trajectories can be compared to, and to emphasize that they come from a different underlying distribution. Even if the internal components of PAULE are pre-trained on segment-based cp-trajectories, PAULE manages to overcome the limitations of the segment-based cp-trajectories. This is possible thanks to the predictive goal-directed nature of PAULE, which focuses on optimizing the planned cp-trajectories for acoustic and semantic targets.

3.2 Acoustic representation

The acoustic representation aims for a psychological, perceptual measure of the acoustic signal rich enough to capture nearly all of the fine details that are needed in speech perception. The acoustic representation approximates the information that the human middle ear sends on to the primary auditory cortex. Therefore, this acoustic representation is not a physical but a perceptual data structure that allows to approximate acoustical similarity.

For computational modeling, it is an advantage that the human ear implements a frequency decomposition and changes a highly oscillatory sound pressure wave into

a frequency decomposed spectrogram. A spectrogram is the time-series of spectra calculated over consecutive intervals of one audio signal. The intervals or windows used to calculate the spectra can have an overlap. The computational advantage of the spectrum is that it is not oscillatory anymore but shows smooth, i. e. highly autocorrelated, transitions between times of high intensity and phases of low intensity for a given frequency band.

Originally, we started with a low dimensional discretized acoustic representation of 21 frequency Mel channels defined for every 10 Milliseconds[4]. While this low-dimensional representation is useful for modeling human perception, it turned out to miss out on the fine details necessary to properly evaluate the quality of speech produced by PAULE. The acoustic representation used in PAULE is much more fine grained. It has a relatively high resolution in the frequency range of 10 Hz to 12,000 Hz and calculates a spectrum every 5 Milliseconds (220 samples of 44,100 Hz audio signal). The frequencies are split into 60 bands on a perceptual Mel frequency scale and the magnitude of the frequency components are logarithmized to approximate the human perception of loudness. The result is a logarithmized Mel banks spectrogram (log-mel-spectrogram) with 60 channels and a time-step of approximately 5 Milliseconds. In a final step, the loudness values in the log-mel-spectrogram are rescaled into the range of zero to infinity. Where a loudness of zero corresponds to silence and a loudness value of one corresponds to a loud clear tone. The scaling factor for the linear rescaling is $1/83.52182518111363$ and intercept is -83.52182518111363 (Listing 3.2).

Listing 3.2: Creating the acoustic representation

```
1  import numpy as np
2
3  from paule import util
4
5  # allocate 5000 "audio samples"
6  audio = np.zeros(5000)
7  log_mel_spectrogram = util.librosa_melspec(audio, 44100)
8  normalized_mel = util.normalize_mel_librosa(log_mel_spectrogram)
9  inv_norm_mel = util.inv_normalize_mel_librosa(normalized_mel)
10 assert np.allclose(log_mel_spectrogram, inv_norm_mel)
```

Note that the log-mel-spectrogram emulates and approximates the human ear, but does not include any higher order processing, taking place in the human brain. Furthermore, it only uses a mono signal and, therefore, no biaural stereo information can be processed by PAULE. In terms of data reduction, the acoustic representation reduces 220 sound pressure values of the mono audio signal to 60 Mel-frequency loudness amplitudes and, therefore, reduces the number of data points by a factor of 3.7.

By creating log-mel-spectrograms, the physical measures of frequency and energy get converted to the perceptual measures of pitch and loudness. Figure 3.1 illustrates how the log-mel-spectrogram is calculated. The transformation for the sound wave to the log-mel-spectrogram is implemented with the help of the Python library `librosa` (version 0.8.0). If the initial signal does not has a sampling rate of 44,100 Hz, it is

resampled to a 44,100 Hz audio mono signal. In PAULE the log-mel-spectrogram is used for three different and distinct purposes: the target acoustics, the predicted acoustics, and the produced acoustics. These three acoustic representations are defined and described in detail in the next three sections and are interdependent on each other.

3.2.1 Target acoustics

The target acoustics defines the acoustical goal, which PAULE tries to mimic by copy-synthesis. It stays the same over the complete planning in PAULE and can be seen as the internal acoustical image that PAULE has in mind for the word PAULE wants to articulate. The target acoustics, therefore, is the reference that is iteratively approximated in the planning. More precisely in each planning step, the Root Mean Squared Error (RMSE) is calculated between target acoustics and the predicted acoustics (see below). The RMSE is a point-wise distance metric, that is calculated by computing the difference loudness for all Mel mean over all data points, and, in the end, taking the square root. It can be seen as a high-dimensional Euclidean distance metric.

3.2.2 Predicted acoustics

The predicted acoustics is the log-mel-spectrogram that is internally predicted by PAULE without synthesizing any sound wave with the VTL. After each iteration of the planning procedure, the newly adjusted cp-trajectories lead to a new predicted acoustics. At each iteration, the predicted acoustics is compared with the target acoustics. Additionally, a predicted semantics is calculated from the predicted acoustics and compared with a target semantics. As the semantics is only calculated from the acoustics, all semantic information needs already be present in the acoustics and no silent speech movements can influence the semantics.

During the planning of the cp-trajectories, many different predicted acoustics are calculated. They can be seen as an internal image of what the articulation would sound like for the current set of cp-trajectories. It is crucial to understand that this is only an internally imagined acoustical image that might differ from the actually produced acoustics from the VTL. The predicted acoustics is used in two ways. Firstly, it is used in an internal loop to plan the cp-trajectories. Here, it is compared to the target acoustics. Secondly, the predicted acoustics is used in an outer loop to further improve the predictive forward model. Here, it is compared to the produced acoustics.

3.2.3 Produced acoustics

The produced acoustics refers to the log-mel-spectrogram that belongs to the audio, which is synthesized by the VTL given some cp-trajectories. The produced acoustics is used to evaluate the overall performance of PAULE. In the outer loop discrepancy between the predicted acoustics and produced acoustics is used to improve the predictive forward model and therefore improves the ability of PAULE to correctly predict an acoustic image from a set of cp-trajectories.

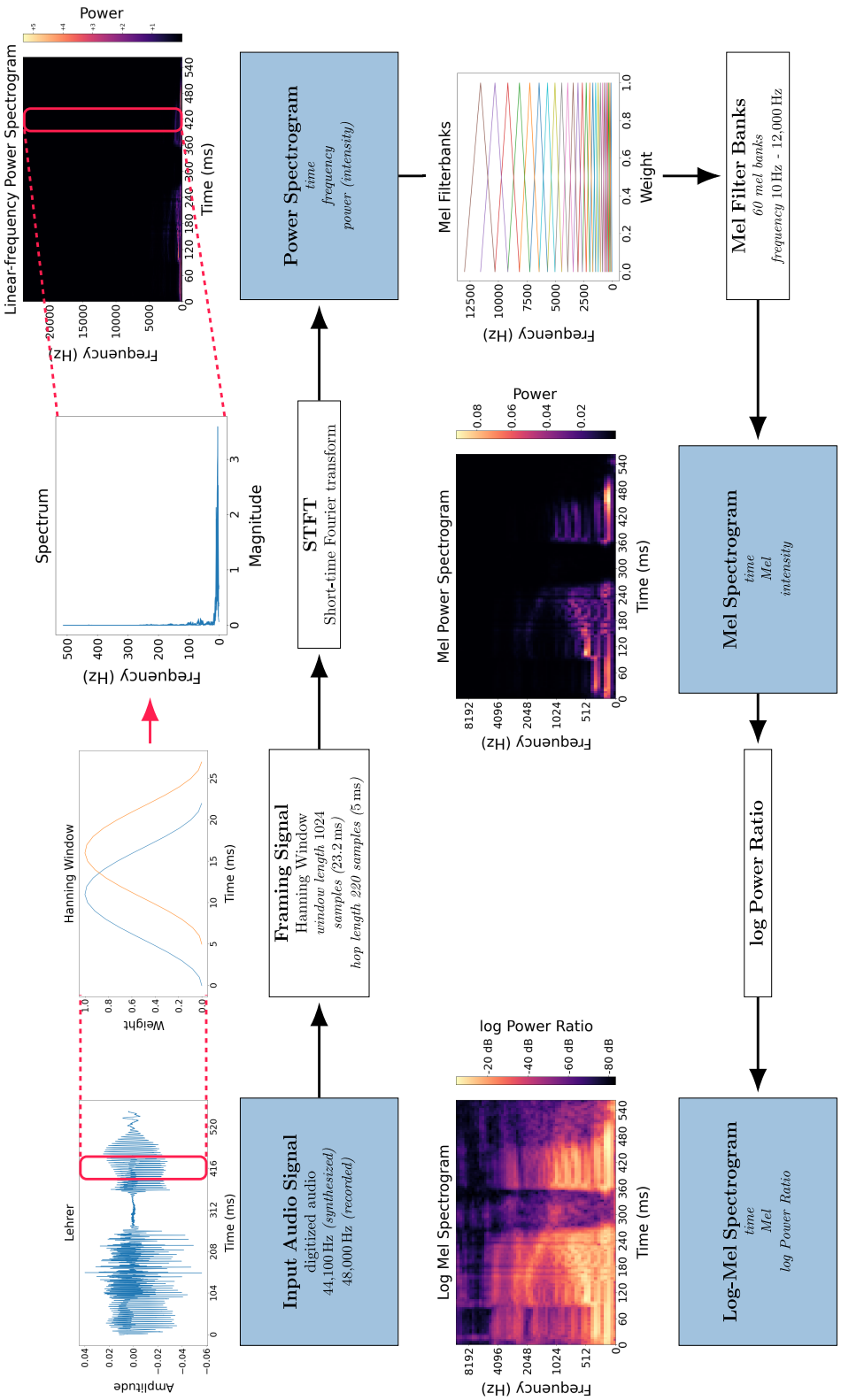

Figure 3.1: Data flow on how a log-mel-spectrogram is calculated. First, the audio signal is split into windows of 1,024 samples with an overlap of 804 samples. These 1,024 samples of the audio signal are then smoothed with a Hanning window and a spectrum is calculated by applying a Short-time Fourier transform (STFT). All of the spectra concatenated along the time axis constitute the power spectrogram. The frequency axis of the power spectrum then is rescaled with a set of triangular filters to transform it to the Mel bank spectrogram of 60 Mel banks between 10 Hz and 12,000 Hz. In a last step the power or energy values are transformed to loudness values by applying the logarithm to them. The implementation used in PAULE is done with the Python library `librosa`.

3.3 Semantic (lexical) embeddings

The data structure for the semantics is a 300-dimensional dense vector space where it is assumed that Euclidean metric holds. As the semantic lexical word embedding vectors (semantic vectors), pretrained fastText vectors[40] are used. These are originally trained on big text corpora more specifically on the common crawl and the German Wikipedia. Each word type is embedded into a dense vector space, where semantically related word types are close to each other and word types of different meaning have a large distance in this space.

As proximity means similarity, these semantic or lexical vector spaces reflect similarity in meaning in the language in a gradual way. Figure 3.2 shows a 2-dimensional simplified version of a semantic space. PAULE relies and uses this gradual embedding space to do word classification and to derive an error signal on the semantic level. PAULE does not alter or change the embedding space, but takes it as a ground truth. Thus, for PAULE a word's location in the fastText semantic space is a numeric representation of its meaning.

fastText vectors are trained on big text corpora and are therefore textual, lexical embeddings. This is not the only way, to create meaningful semantic embeddings. As fastText only relies on text, acoustic similarities should not be (strongly) reflected in the embedding space. In contrast, embeddings build on top of large speech corpora should reflect acoustic similarities together with the cooccurance patterns and predictive value, which is also present in the fastText vectors.

Recent research puts some focus on combining different embedding methods and creates multi-model embedding vectors that combine pictures and text[99]. The PAULE version here is trained and tested on the 300-dimensional fastText vector space, but given some training time it is trivial to change the semantic representation to a different embedding space as long as an informative error can be defined between a predicted semantic vector and a target semantic vector.

Besides dense semantic embeddings, sparse embeddings like the one used in Sering et al.[90] are possible as well, but might need a different metric defined on them. Especially, for the case of a sparse binary meaning space, the Euclidean metric and the associated RMSE loss might not be the right choice. Furthermore, metric vector spaces assumes that these embeddings have a meaningful inverse element and that each vector is scalable. These two core assumptions of the metric vector space definition are questionable as many embeddings are created to have a confined length of the vector and large volumes of the vector space are empty and there is no clear interpretation of the inverse. One solution might be in the use of manifolds and differential geometry to adequately model dissimilarity of word meaning structure. Unfortunately, this is out of the scope of this thesis. In the PAULE framework approximating and assuming that the semantic vectors are elements of a metric vector space turns out to be quite useful. In the end, PAULE relies mostly on the local structure and proximity of vectors and not so much on the global correctness of the metric in the vector space.

There are three distinct and important semantics in PAULE that each consists of a 300-dimensional vector that live in the same vector space.

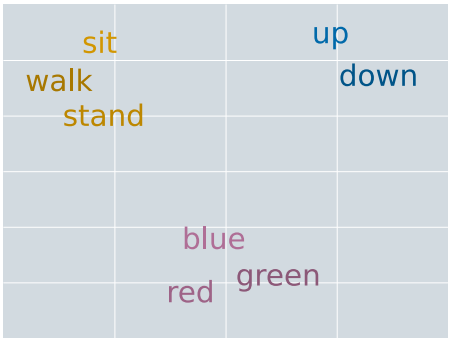

Figure 3.2: Sketch on how semantic embeddings can be conceptualized. The semantic embedding space used by PAULE is not 2-dimensional, as in the sketch, but 300-dimensional.

3.3.1 Target semantics

The target semantics is a 300-dimensional fastText vector that either is selected by the user, who uses PAULE to synthesize speech, or is the associated semantics to a target acoustics. The purpose of the target semantics is to inform PAULE as to which semantic vector the resulting audio speech waveform should be embedded.

As many different waveforms (around 22,000 data points per half a second of speech) are embedded on the same or similar 300-dimensional vector, the task to find an appropriate waveform involves another inverse problem. This inverse problem is mainly mitigated by the fact that the waveform needs to contain synthesized speech from the VTL, which is further constrained to be speech form the *JD3* speaker geometry. It is difficult to quantify the dimensionality reduction due to this constraint, but it is clear that first, it is huge and second, it is not as small as the 300-dimensions of the semantic vector space.

Additionally, it is intended that different sounding words like /Bahn/ (train) and /Zug/ (train) map onto similar semantic vectors. Therefore, if a specific acoustic version of a word should be uttered, besides the semantic target an acoustic target might be needed.

The target semantics is used as a reference for the predicted semantics in the internal loop and a semantic error is derived from the RMSE between those two. For one synthesis, the target semantics is constant. The target semantics is used in the same way as the target acoustics.

3.3.2 Predicted semantics

The predicted semantics is the semantic 300-dimensional fastText vector that is associated with the predicted acoustics. The predicted semantics is used in the internal loop and can be interpreted as the internal imagination of what the meaning of the planned articulation of the target word is. This is similar to the predicted acoustics and as in the predicted acoustics, the articulatory synthesizer VTL is not involved in creating the predicted

semantics. Therefore, the predicted semantics is a purely internal representation. This predicted semantics than can be compared to the target semantics and the error between the prediction and the target is used to further improve the cp-trajectories in the internal loop.

3.3.3 Produced semantics

In contrast to the predicted semantics, the produced semantics is the semantic 300-dimensional fastText vector that is associated with the finally produced acoustics by giving the planned cp-trajectories to the VTL synthesizer.

The target semantics is used to evaluate the overall performance of PAULE. If the produced semantics is close to the target semantics, the articulatory planning worked as intended and the audio wave produced by the VTL synthesizer should activate the corresponding meaning both in computational models of human auditory comprehension (see, eg.,[98]) as well as for human listeners.

4 Data sets

> Big Data is like teenage sex: everyone talks about it, nobody really knows how to do it, everyone thinks everyone else is doing it, so everyone claims they are doing it.
>
> Dan Ariely

The data structures introduced in the last chapter are connected along different data sets. In this chapter, the data sets as well as the preprocessing is introduced. The data sets and their properties are important as they instantiate the data structures and therefore provide exemplars of the data structures and the association between the data structures. These exemplars, also called training samples or learning events, can be used to train machine learning models (or to fit statistical regression models) with supervised or semi-supervised learning. Machine learning models are used as the internal models of the PAULE framework. The internal model definitions and implementations are discussed in the PAULE chapter (Chapter 5).

An important precursor and contribution of this thesis was the joint development of a pipeline to create large amounts of training data for machine learning models operating on the VTL[94]. This work was then further refined by Yingming Gao and integrated into VTL in version 2.3. Now this segment-based approach can be used by everyone to synthesize speech from sequences of phones and duration. Before this joint contribution, the resynthesis of aligned speech corpora could only be done in a semi-automatic way and, therefore, was very time-consuming even for the number of training samples with no more than 100 exemplars. In our contribution, we developed a way to efficiently generate training data with 50,000 exemplars and more.

In language and speech research, especially when cognitively motivated, it is important to be able to generate large amounts of training data to capture the distributional properties of a language. An important distributional feature of language – in written as well as in spoken language – is the strong class imbalance called Zipf's law. Figure 4.1 shows the number of word types (word classes) in the Mozilla Common Voice corpus (Common Voice) corpus[1] (introduced in a following section) sorted by the most frequent one on the left to the least frequent word type on the right. The structure within the plot is not really visible as the most frequent class has around 1,113 occurrences, whereas the majority of the word types (2,261) occur only once. A common way to visualize such distributions is to plot rank and frequency in a double logarithmic plane (Figure 4.2). A straight line in the log-log plot indicates a power law. Zipf's law states that word types follow a power law[118]. If Zipf's law is true, is still under debate, but it is commonly

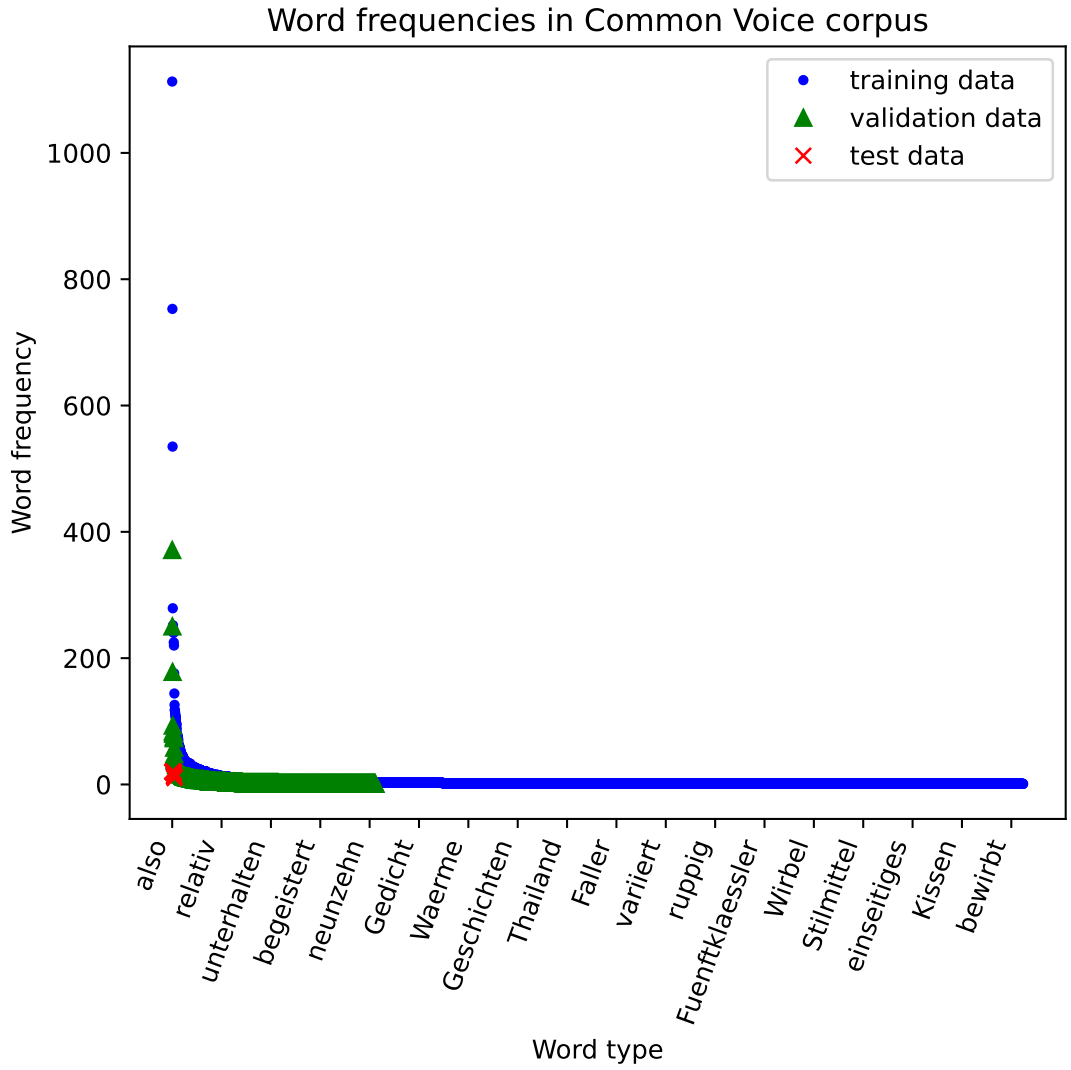

Figure 4.1: The number of tokens (occurrences) per word type (class), sorted so that the most frequent word types are plotted on the left and the least frequent word types on the right. The most frequent word type /also/ occurs 1,113 times in the Common Voice training data. There are 2,261 word types in the training data that occur only once. The total number of word types is 4311. In addition to the training data the validation and test data sets are shown. These share a subset of the word types, but consist of word tokens that are not in the training data.

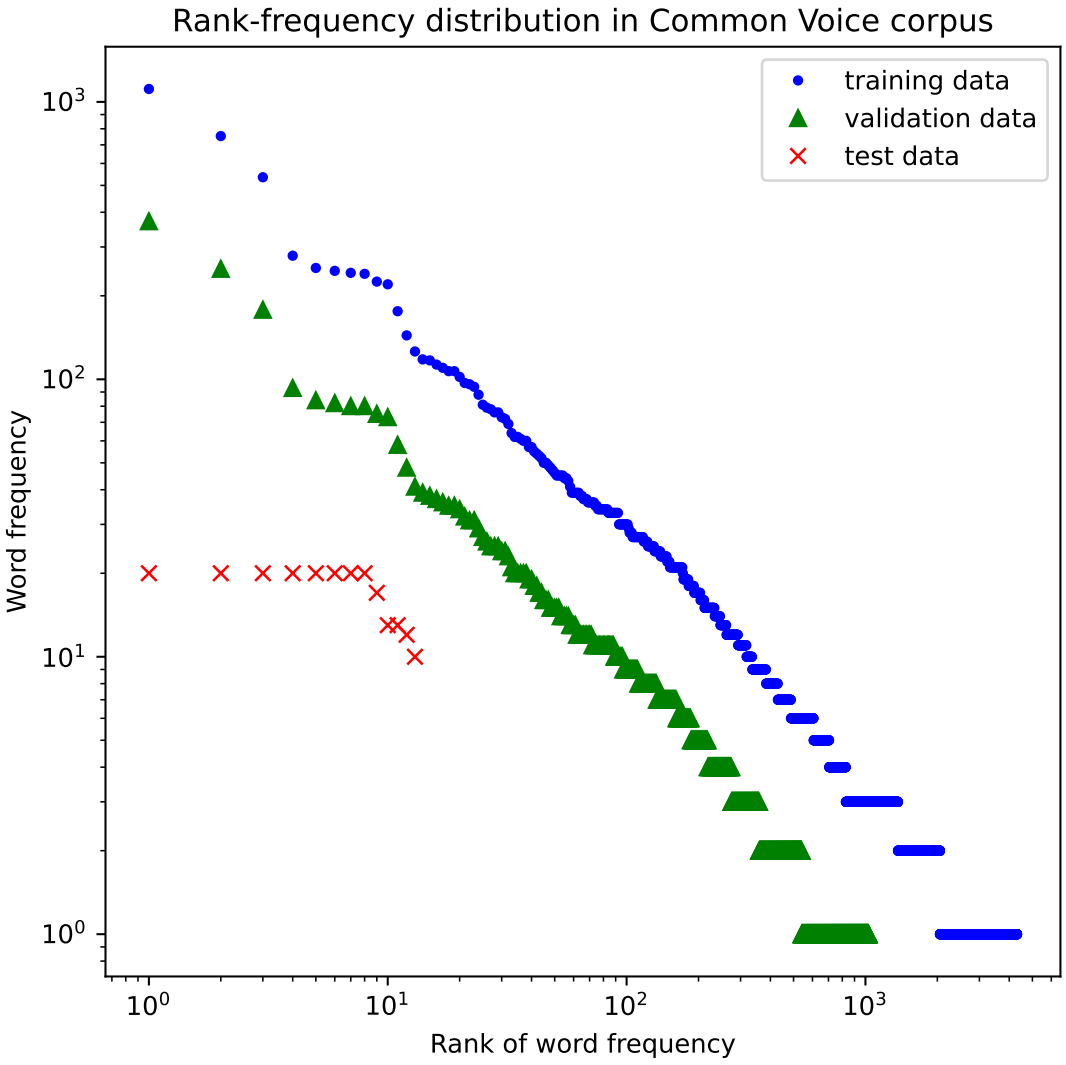

Figure 4.2: Rank-frequency distributions for training, validation, and test data of the Common Voice dataset. Except for the test data, the rank-frequency distributions are roughly approximated by a power law.

excepted that there is a strong class imbalance, i. e. relatively few high frequency word types and many word types with very low frequencies[6].

Another important statistical aspect of speech is that high frequent word types show a larger variability and larger coarticulation and more idiosyncrasies compared to low frequent word types that are more regular, less variable, and less coarticulated[54,28]. With the segment-based approach, this aspect is partially reflected as the duration of the phones from the training data are used and respected by the pipeline. But the systematic differences in coarticulation between high and low frequent words are not reflected. For a discussion of the relation between frequency of use and spoken word duration from a cognitive perspective, see Gahl[37]. In that respect, PAULE is expected to improve this aspect of speech synthesis on the segment-based approach.

4.1 Segment-based Synthesis and the GECO corpus

In the 2019 contribution, we resynthesized the GECO corpus[84], which contains conversational speech recorded in the laboratory between female southern German young adult speakers. The resynthesis resulted in 53,960 spliced out word samples to a total of 6 hours and 23 minutes of synthesized speech. The speech recordings are annotated on the word and phone level so that individual word tokens can be extracted from the continuous recordings. The phone alignments can be used in the segment-based synthesis approach to generate large amounts of training data.

Our initial pipeline is depicted in Figure 4.3. It works in the following way: As inputs, the segment-based approach receives a sequence of phones in SAMPA transcription. For each phone, the duration of that phone is available as well. For each phone that exists in the German language a gestural score is defined. This gestural score was manually predefined by the authors of the VTL and was informed by MRI scans and expert knowledge on articulation.

The gestural score defines target positions and timings for the 30 control parameter. Together with the durations these are connected and blended together and result in smooth cp-trajectories. As no intonation is encoded, the control parameter for the F_0 contour – the pitch of the speech signal – is flat and has no variability. Therefore, in a second step, we extracted pitch contours from the recordings in the GECO corpus, using Praat[17]. For each word token we fitted a smooth curve to the extracted pitch with the TargetOptimizer[79] software. The resulting smooth pitch curve was added to the gestural scores. As a result, the resynthesized input F_0 follows the F_0 of the original recording.

The segment-based synthesis outputs one set of cp-trajectories that can be synthesized with the VTL. For the synthesized audio, the log-mel-spectrogram and the semantic vector can be calculated, and, therefore, triplets of training data are generated: the input cp-trajectories, the acoustic representation, and the semantic representation.

Note that this segment-based synthesis model, in contrast to PAULE, uses symbolic units in acoustics in the form of phonemes and at the motor level in form of gestural scores. Creating gestural scores for all phonemes of a language requires a lot of handcrafting and

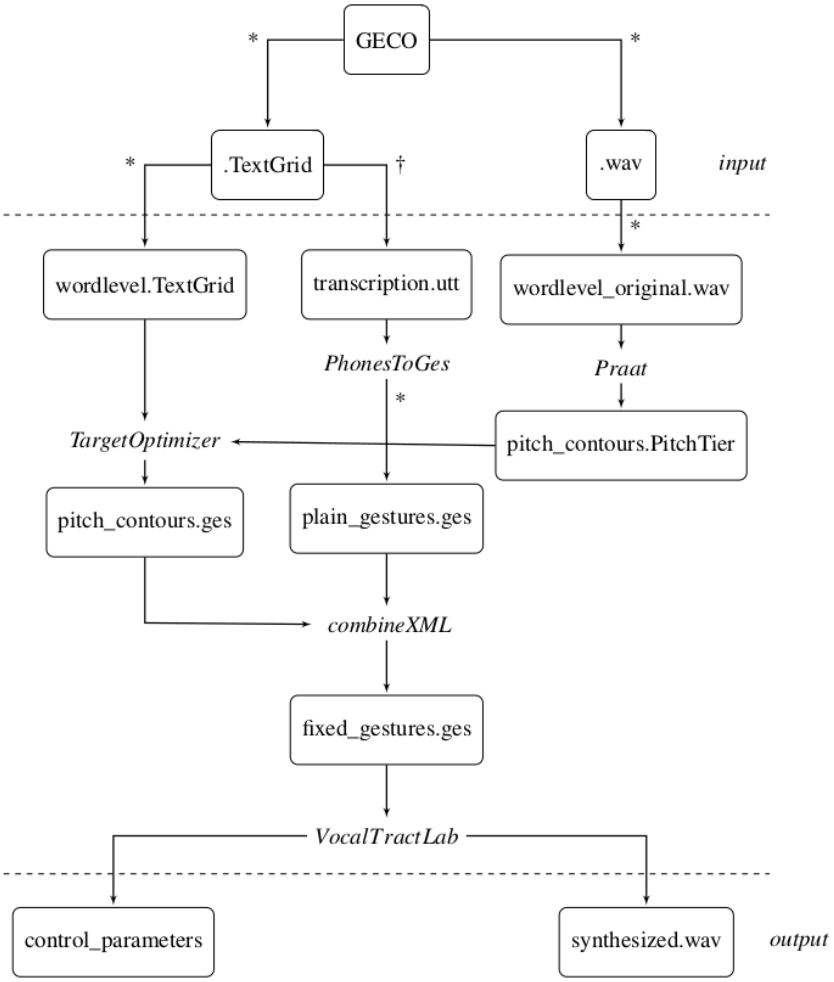

Figure 4.3: Segment-based synthesis pipeline for the VTL speech synthesizer. This is used as a reference model and to generate large amounts of training data for PAULE.

expert knowledge. As we use these handcrafted gestural scores, this expert knowledge is leaked into the PAULE framework through the initial training procedure. In the discussion, I will briefly discuss approaches for which no initial training data is needed and all the experience is built up from zero knowledge in an exploratory scheme.

To make the segment-based synthesis model comparable to PAULE we added a preprocessing pipeline that allows to use the segment-based synthesis model on target audio recordings. For this a target audio recording is first classified by a word classifier to derive its content word type. In a second step a phone transcription of the predicted word type is aligned to the target audio. From the alignment the phone durations can be extracted so that the segment-based synthesis model has all information it needs to do a resynthesis. This preprocessing pipeline allows to compare the segment-based synthesis model to PAULE. Furthermore, this pipeline with the segment-based synthesis can be used as an alternative way of initializing PAULE and PAULE can be used to refine the copy-synthesis.

4.2 Montreal Forced Aligner (MFA) and Mozilla Common Voice

Unfortunately, the GECO corpus is only available to scientists and cannot be freely distributed. In order to ensure general replicability, we resynthesized another training set out of the freely available Mozilla Common Voice corpus (Common Voice)[1]. The Common Voice data set is a crowd-sourced collection of read-out sentences. In contrast to the GECO corpus, the audio in the Common Voice is neither spontaneous speech nor is it aligned at the word or segment level. Still, it is spoken by normal people in diverse environments with consumer electronic microphones. This makes the audio recordings as ecological valid as studio recordings or captures of TV or radio broadcasts[51], but less ecological valid compared to the GECO corpus. To ensure that the transcription of the sentences are correct all sentences are checked by crowd-sourced listeners.

The Common Voice makes recordings available in many languages. For the purpose of this thesis, we focus on 26,271 word tokens distributed over 4,311 word types of the German recordings, which is only a small part of the German Common Voice. We selected those sentences that contained most of the words present in the GECO corpus. This way, we created a data set similar to the GECO, which allows for tests of transfer learning, i. e. how well a given model architecture performs when it first was trained on the GECO data and then evaluated on Common Voice. In a word classification task, i. e. predicting the word type from the audio, very limited transfer was possible between GECO and Common Voice and word classification accuracies dropped from 60 % to below 5 % on the spliced-out audio recordings.

Before any models can be trained on Common Voice, the transcriptions and audio recordings first need to be aligned, i. e. the word boundaries have to be identified and audio tokens have to be assigned to word types. Furthermore, to use the segment-based approach, phones need to be defined for each word and aligned within each word so that that the phones and their durations can be made available to the segment-based approach.

Properly aligning transcriptions with audio recordings on the word and phone level with high quality would involve substantial amounts of work from a well-trained phonetician. As we were mainly interested in having enough variability in our training data, we could be lax on the quality of the alignment as long as the variabilities in the durations resemble the statistics in the real language and the audio tokens are correctly aligned to the word tokens. As we allowed for this big error margin, we could fully automate the process of alignment. We validated the automatic alignment on a small portion of the data manually. Furthermore, we inspected all the data that either was very long compared to the rest of the data, was consistently misclassified by our classifiers (the embedding models described in Section 5.4), or showed some other odd behavior. Wrong samples were then removed from the data set. Overall this way only around 100 samples were removed out of 27,000.

To do the word and phone segment alignment we used the Montreal Forced Aligner (MFA)[63], a freely available and FOSS aligner that has pre-trained weights available for many different languages including German. In a first run, Montreal Forced Aligner (MFA) takes the audio of the full sentence and the text transcription as input and outputs the word boundaries. These are used to split the audio into chunks that only contain a single word. These single spliced-out words still contain coarticulation patterns and carry information from its surrounding and are substantially different from recordings of single read-out words. In the last step, a phone transcription of the target word is looked up and in a second run the phones are aligned to the spliced-out audio recording. Besides generating another training corpus for PAULE, this pipeline makes it possible to create new training corpora for other languages with relative ease, as long as the gestural scores for all the SAMPA phone transcriptions are available or can be proxied by similar sounding phones.

4.3 Electromagnetic Articulography (EMA) and the Karl-Eberhard Corpus (KEC)

One of the big benefits, if not the big benefit, of an articulatory speech synthesis system, is the ability to compare it to human articulation. A widely used technology for measuring articulation is electromagnetic articulography (EMA) (figure 4.4). Sensors are glued to the tongue, lips, and head. These sensors can be tracked in an inhomogeneous magnetic field that is projected through the head of the participant. As weak magnetic fields are harmless and only very weakly interact with the head, it is possible to measure the movements of the tongue while it is in the mouth with a high resolution in time of up to 1,000 Hz. The system we used runs with 400 Hz, which is enough for most fast articulatory movements. Limitations of EMA are the limited spacial resolution, the long preparation times and the limited head movement of the participant. Furthermore, most participants need to first accommodate to the bundle of small electrical cables coming out of their mouth, which slightly interferes with the speaking process.

One German corpus that contains EMA recordings of spontaneously spoken southern

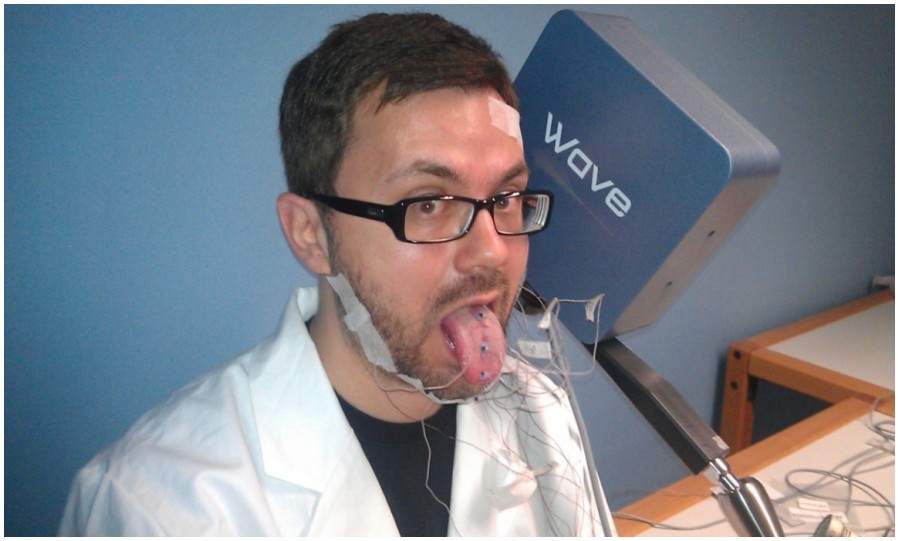

Figure 4.4: Small sensors on wires are glued to the tongue and can be tracked in the 3-dimensional space in front of the blue box, which emits an inhomogeneous magnetic field. The time resolution for this setup is around 400 Hz, but systems are available that have a time resolution of 1,000 Hz. The electromagnetic articulography (EMA) system, therefore, is capable of measuring the fast movements of the tongue in the mouth while speaking.

German dialogues is the Karl Eberhard Corpus (KEC). Besides having EMA recordings available, the KEC is aligned and manually validated at the word level and automatically aligned at the phone level, which makes all information available that is required for segment-based synthesis.

To compare EMA recordings with the VTL-based synthesis, I developed together with Hannah Schütt an automatic way of extracting virtual EMA trajectories from cp-trajectories. The code changes introduced in this student project allow to trace movements of any node of the 3-dimensional mesh that defines the geometry of the VTL. A preset of common places of virtual EMA points are available. A first pilot study demonstrated that it is possible to extract virtual EMA points from VTL and compares these to human EMA recordings extracted from the KEC[95].

4.4 Ultrasound recordings and /babibabubaba/

Another technique that is widely used to study articulation is ultrasound. A recording device is strapped to the chin and records the contour of the upper side of the tongue body. The system used in this thesis records tongue movements in the mid-sagittal plane at 80 Hz. Figure 4.5 shows an example measurement. The white line is the reflection of the ultrasound signal between the tongue blade and the air in the oral cavity.

As with the EMA data, we made it possible to automatically extract height information

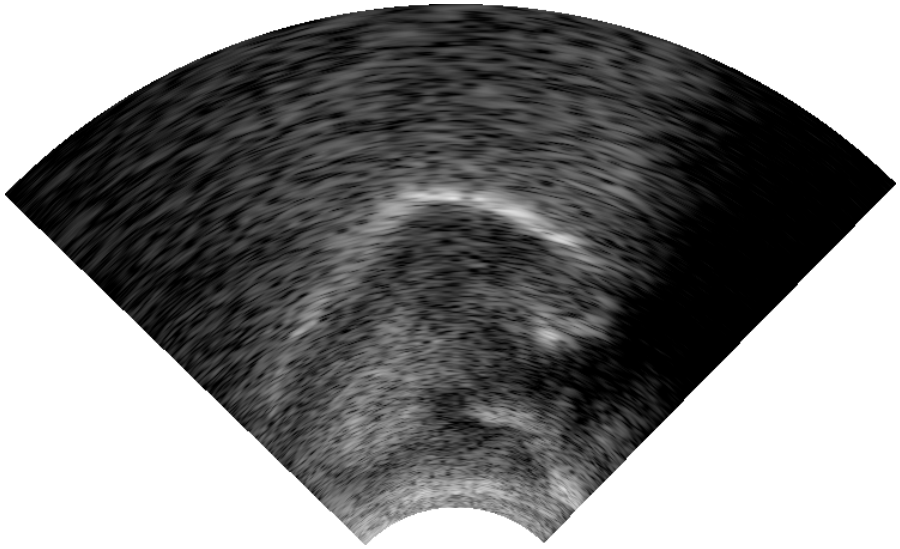

Figure 4.5: The fan-shaped recording of the mid-sagittal plane of the tongue with ultrasound. The brighter pixels are visible where the tongue muscle ends and the air of the oral cavity starts.

of the mid-sagittal plots of the VTL. This allows us to quantitatively compare the measures from the ultrasound recording to the virtual measures extracted from a VTL synthesis. To show this ability we recorded in our lab the utterances /baba/, /babi/, and /babu/ at different speaking rates and with varying number of repetitions. These German non-words give rise to strong anticipatory coarticulation, which is visible in the mid-sagittal plane. This coarticulation effect systematically affects average tongue height as well as the variability in the first /a/. Furthermore, it results in systematic changes in the vowel quality at the end of the first /a/, which can be quantified with the formants (resonance frequencies) f1, f2, and f3. In contrast to the segment-based approach, PAULE successfully modeled the formant shifts that are the result of vowel-to-vowel coarticulation[96].

First steps have been done to combine all the data sets and analysis mentioned here into a benchmark that can be automatically run on control models of the VTL. This benchmark is named *articubench* and is described in[91]. In Section 6.6, more details on articubench are given.

5 Predictive Articulatory speech synthesis Utilising Lexical Embeddings (PAULE)

Wer ist eigentlich PAUL(E)?

AFTER A FAMOUS GERMAN COMMERCIAL

The main contribution of this thesis is the Predictive Articulatory speech synthesis Utilizing Lexical Embeddings (PAULE) framework, which connects the different data structures defined in the chapters before and is a control model for the VTL. PAULE is predictive in the sense that it internally predicts or imagines the acoustic manifestation (log-mel-spectrogram) of not yet executed articulatory movements (cp-trajectories). It then maps these internally imagined acoustics into a lexical embedding space (semantic vector). The imagined acoustics and corresponding semantics are compared to a desired target acoustics and target semantics. This comparison yields an error that is used to correct the not yet executed cp-trajectories (see Figure 5.1).

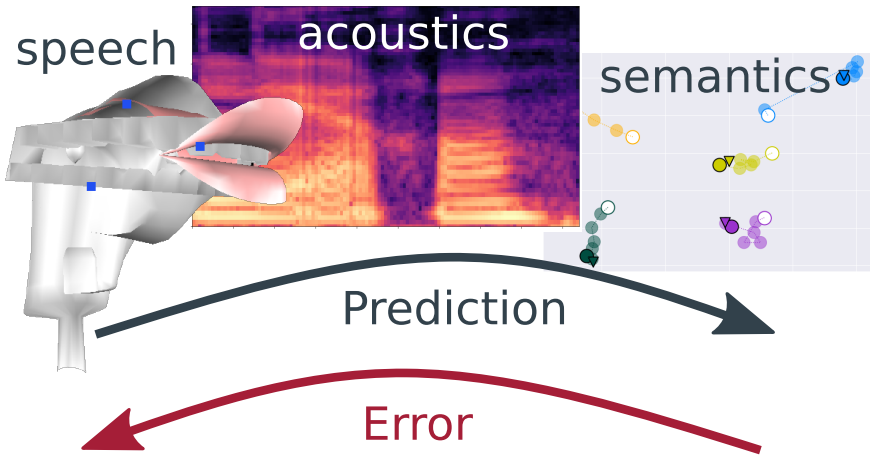

Figure 5.1: The main idea of PAULE is to predict the acoustics and semantics of the upcoming articulatory movements (cp-trajectories) and use the error between the intended target acoustics and the intended target semantics to plan and correct the movement.

The PAULE framework derives articulatory movement trajectories and corresponding glottis parameters of the articulatory speech synthesizer VocalTractLab (VTL). It does not use any symbolic representation, neither a phone-like representation on the acoustic

side nor any gestural scores or movement targets on the motor side. The cp-trajectories of the VTL speech synthesiser derived by PAULE can be obtained for a target sound file (copy-synthesis), for a target semantics (semantics to speech) or for a combination of both. Resulting cp-trajectories fulfill the physical soft constraints of locality (only move an articulator to a new position if you have to) and constant force (if an articulator is moved, move it with constant acceleration or force). This leads, in most cases, to human-like movement trajectories. In the version presented here, PAULE only produces word tokens. Deriving cp-trajectories for a word with a duration of half a second requires roughly 30 minutes of computation time. With these properties, PAULE fulfills the goal set for this thesis and follows the guiding principles described in Section 1.1. PAULE is available as a python package from PiPy or from `https://github.com/quantling/paule`.

5.1 Journey to PAULE

Even with simplicity in mind, the PAULE framework turned out to be rather complex. To fully understand PAULE and see the beauty in Figure 5.9, this section builds up to the full framework in a stepwise manner.

The journey starts with Figure 5.2. As described and defined in the last chapters, a target semantics and a target acoustics are present as well as the VTL. The VTL receives cp-trajectories as inputs and outputs synthesized audio as a 44,100 Hz mono signal. This audio signal can be transformed into the acoustic representation of a log-mel-spectrogram.

The first big question that PAULE will answer is how to find suitable cp-trajectories for a given target acoustics. This is the so-called speech inversion or copy-synthesis task. Note, that this is a one-to-many mapping. Many different cp-trajectories map onto similar or the same acoustic representation. At the same time, many different acoustic representations cannot be articulated, i. e. it is not possible to perfectly mimic the speech of a different speaker. This impossibility arises from differences in the vocal tract geometry and the resulting fundamental differences in resonance frequencies. The copy-synthesis therefore has to find a close approximation of the target sound and select one of the cp-trajectories that fits the articulatory movement context.

In Figure 5.2, the semantic representation is present without any connection to other modules yet, but will be connected in subsequent versions of this Figure. In all figures that lead to the PAULE framework, the semantic level will be drawn in the top row of the figure. In the middle row, the acoustic representations and the cp-trajectories are drawn and in the bottom row, the VTL and audio wave signal are shown.

As a first step an error is introduced (see Figure 5.3). More specifically, the Root Mean Squared Error (RMSE) between the produced acoustics as a log-mel-spectrogram and the target acoustics as a log-mel-spectrogram is computed and used as the acoustic error. The RMSE is a point-wise error that is computed on the loudness difference in each time-step Mel-frequency combination.

With the error between the target acoustics and the produced acoustics, a gradient-free optimization method could already be applied. The loop from the cp-trajectories over the VTL to the produced acoustics and followed by a comparison with the target acoustics

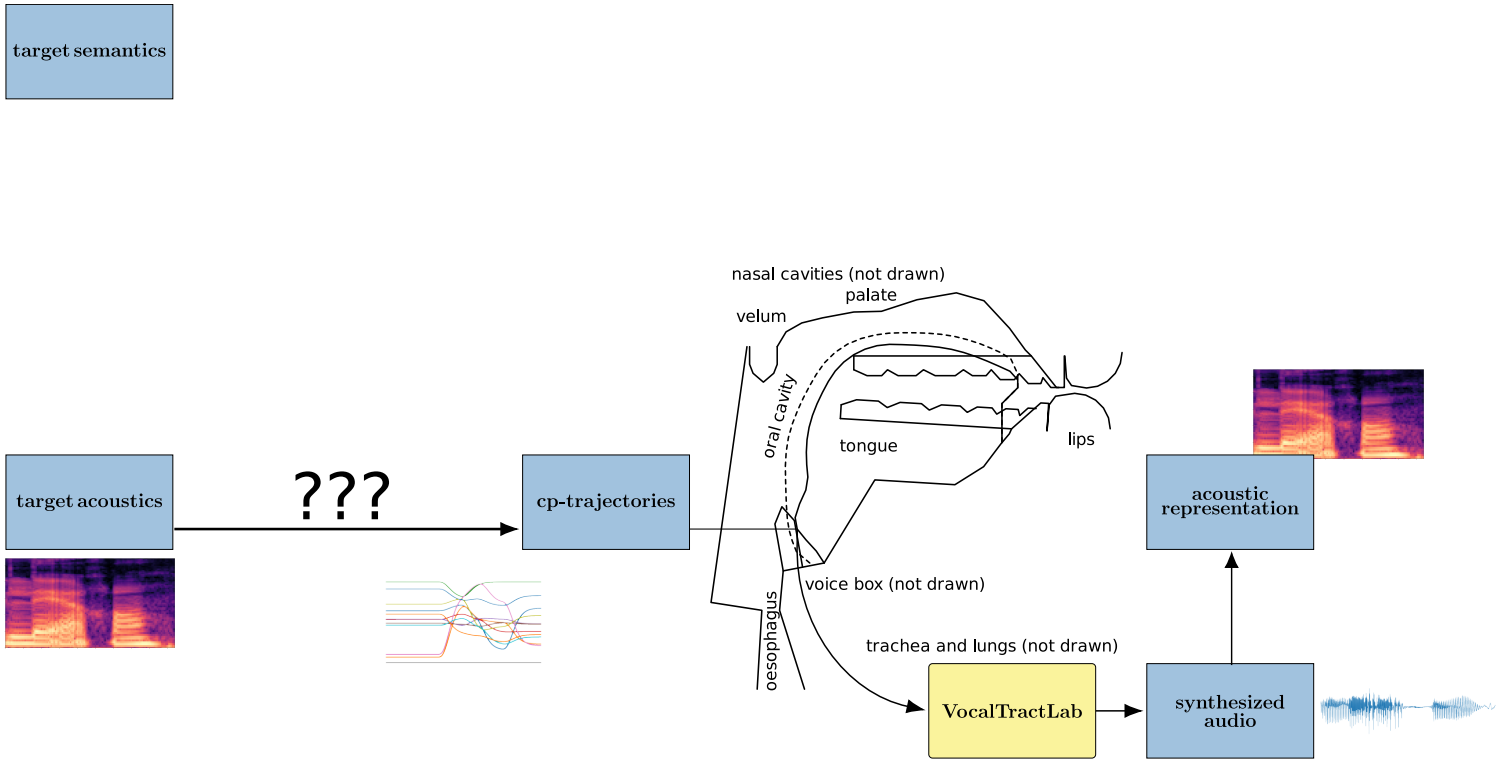

Figure 5.2: Step 1 in the journey to PAULE. The question marks highlight the key challenge of the PAULE framework, which is to derive suitable cp-trajectories for a target acoustics. The cp-trajectories define a time-series of internal states of the articulatory speech synthesis system VTL. With the cp-trajectories the VTL synthesizes a 44,100 Hz mono audio signal. From the audio signal, the acoustic representation can be computed. The acoustic representation and the target acoustics share the same data structure, namely, they are implemented as a log-mel-spectrogram. The target semantics is still unconnected to the cp-trajectories and the VTL and will be connected in a subsequent figure.

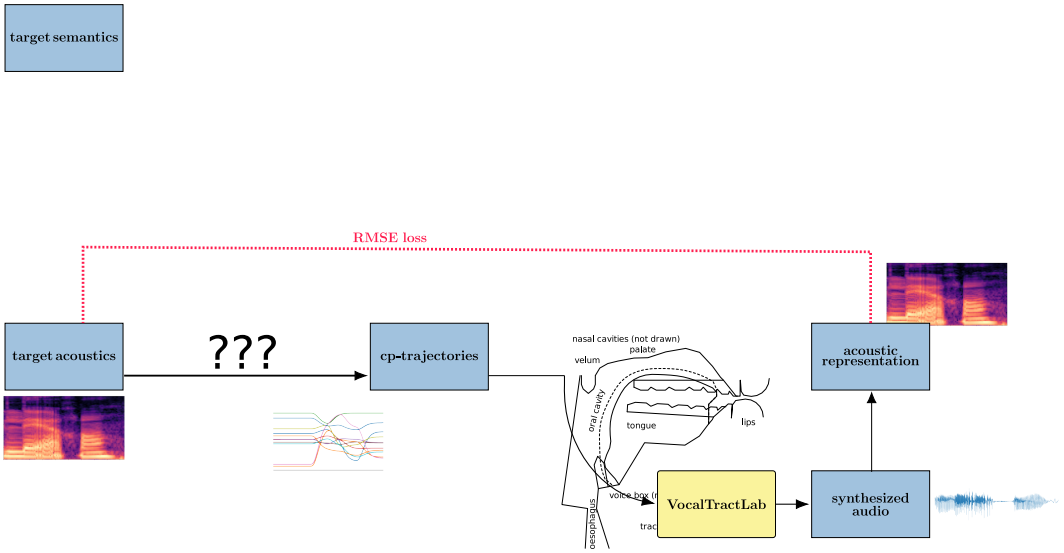

Figure 5.3: Step 2 in the journey to PAULE. As the target acoustics and the acoustic representation share the same data structure, a point-wise error can be calculated that quantifies the difference between the resulting acoustic representation and the target acoustics. For the error calculation, the Root Mean Squared Error (RMSE) loss is used. In this setup, the connection between the target acoustics and the cp-trajectories already could be solved with a gradient-free optimization method like Covariance Matrix Adaptation Evolution Strategy (CMA-ES) purely in the acoustic domain. However, that the high dimensionality of the representation renders naive gradient-free methods in this setup useless or comes with extreme computational costs.

is called the outer loop in acoustics. As we are not only interested in acoustics but do believe in the importance of semantics, we integrate semantics into the outer loop. In order to do this, two problems need to be solved, both of which are depicted in Figure 5.4.

The first problem is how to get from the acoustic representation to a semantic representation. In this thesis, the semantics is derived purely from the acoustics. Therefore, the only way to infer a word's the meaning is by listening to it's audio signal. Especially, there is no lip reading or somatosensory feedback in the present version of PAULE.

The second problem concerns the inverse problem of how to find suitable cp-trajectories for a given target semantics. This is even more of a many-to-one problem, as the semantic representation has a dimensionality that is much smaller than that of the acoustic representation. Therefore a small region in the semantic space needs to map on many different cp-trajectories in the very high dimensional motor space. An example are synonyms, which are close in the semantic space, but have very different cp-trajectories. The inverse problem needs to be able to deal with this structure properly.

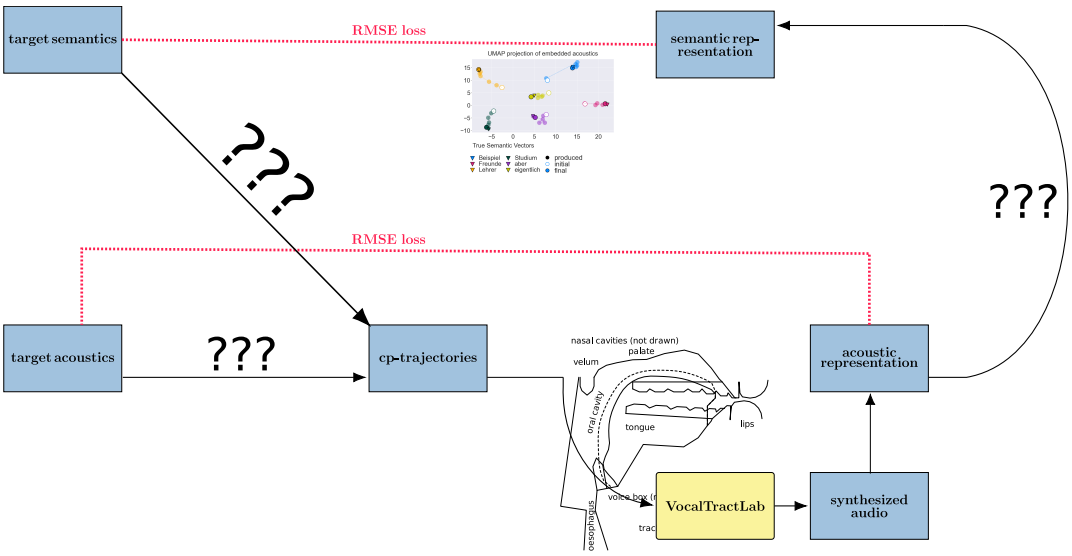

Figure 5.4: Step 3 in the journey to PAULE. Now the target semantics is connected to the cp-trajectories with question marks, which indicates another inverse problem. Furthermore, the acoustics is connected with a yet unknown mapping to a semantic representation. The semantic representation is derived purely from the acoustic representation and has the same data structure as the target semantics.

What solution we propose for the inverse problem and how this solution works is discussed later. But as we have a many-to-one problem in the acoustic to semantic mapping, we can approximate this mapping with a machine learning model and learn this association by giving enough training data. The mapping from the acoustic log-mel-spectrogram to the semantic vector will be called the *embedder* as the model embeds the log-mel-spectrogram into a fastText embedding space. Figure 5.5 shows the outer

loop with the embedder and the RMSE between the target semantics and the produced semantics.

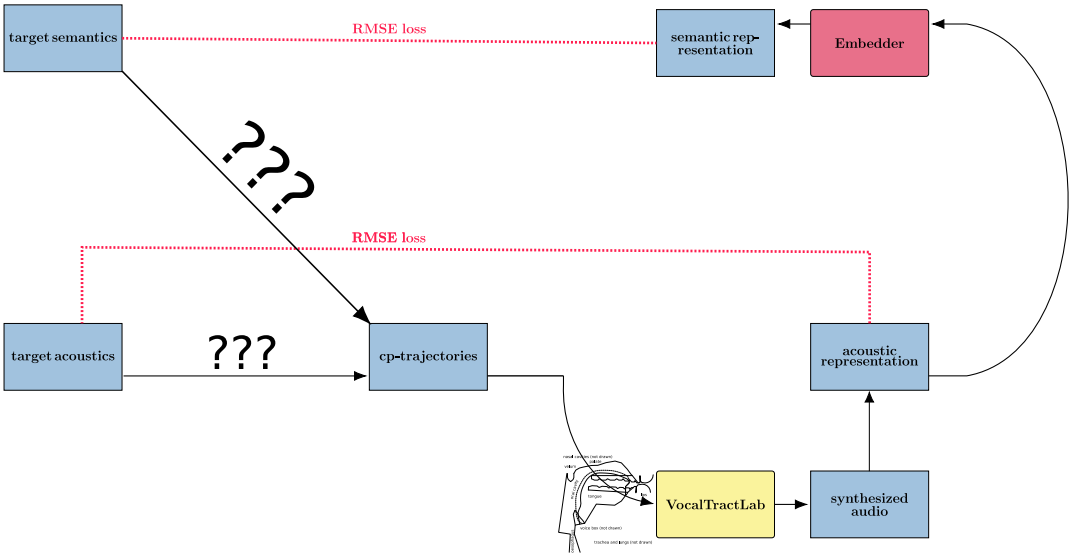

Figure 5.5: Step 4 in the journey to PAULE. Now, a semantic representation is derived from the acoustic representation with an embedding model (the embedder). The embedder is a time-series to fixed vector mapping, implemented as an artificial neural network model. As the acoustic representation is up to here always the result of the synthesis of the VTL, these setups are called the outer loop.

The outer loop shown in Figure 5.5 is a minimal outer loop that combines acoustics and semantics with an articulatory synthesis system. Three tasks can be defined on this outer loop. First, a acoustic-only task (copy-synthesis task) is defined by giving a target acoustics, deriving the associated target semantics through the embedder, and then finding cp-trajectories that match the target acoustics and the derived target semantics as closely as possible. Second, in the semantic-only task, a target semantics can be given and now the task is to find cp-trajectories that lead to an acoustics that has a meaning that matches the target semantics. Here, with the setup displayed in Figure 5.5, no target acoustics is present and no error from the acoustics can be derived as there is no way to go from the semantic vector to the target acoustics, yet. Later, we add in a connection from the target semantics to target acoustics using a Generative Adversarial Network (GAN). The third and last task, the semantic-acoustic task, is to give matching target semantics and target acoustics. In this task, the semantic and acoustic targets are given therefore no target has to be derived indirectly. The third task can be interpreted as repeating the name of an objected that is pointed at (the target semantics) and match the pronunciation as closely as possible to the one given by the teacher (the target acoustics).

For a gradient-free method like an evolutionary algorithm, the semantic and acoustic RMSE would be used as a fitness or scoring function. The evolutionary algorithm would then try to minimize the error by creating populations of cp-trajectories, where in each

generation of the population only a small sub-sample of the best or fittest individual cp-trajectories are used to breed or create the new population of the next generation.

Naively optimizing the outer loop with a gradient-free method like Covariance Matrix Adaptation Evolution Strategy (CMA-ES) has several disadvantages. The fitness function is defined on the whole word, therefore, CMA-ES cannot take advantage of the time and frequency structure of the errors. With CMA-ES, it is difficult to implement accumulation of experience and the CMA-ES needs to articulate and synthesize hundreds and thousands of cp-trajectories for it to (hopefully) converge to optimal cp-trajectories for a given target acoustics and target semantics. A last notable challenge is that producing silence might be favorable in terms of low RMSE in the acoustic representation compared to audio synthesized by intermediate cp-trajectories. In this silence the CMA-ES get stuck as to create cp-trajectories that are favored over the cp-trajectories that produce silence is practically impossible due to the high dimensionality of the cp-trajectories.

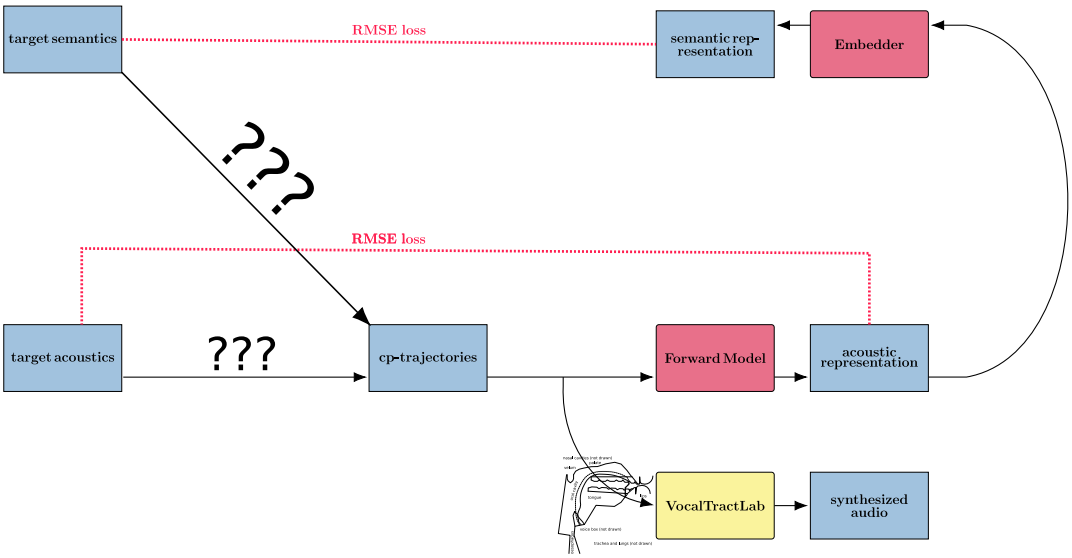

Figure 5.6: Step 5 in the journey to PAULE. Now, the outer loop is replaced by an internal loop. For the internal loop, the acoustic representation is no longer derived from the synthesized audio, but is now predicted by the predictive forward model, which bypasses the VTL and directly predicts the acoustic representation from cp-trajectories. Two advantages of the predictive forward model are that it executes faster compared to the VTL and that it incorporates experience and, therefore, can be different for an English version of PAULE compared to a German version.

In order to avoid these problematic aspects of evolutionary algorithms, we introduce a predictive forward model. The predictive forward model bypasses the VTL and directly learns the association between the cp-trajectories and the acoustic representation. It, by design, does not model the audio wave, but directly the acoustic percept. Figure 5.6 clarifies how this predictive forward model is integrated with other modules of the PAULE framework. Importantly, the connection between the synthesized audio and the

acoustic representation is removed. The loop shown in Figure 5.6 now is a purely internal loop: no articulation and no invocation of the VTL synthesizer is involved. The internal loop is the same as the outer loop with the exception that the VTL synthesizer is replaced by the predictive forward model and the audio signal is skipped (Figure 5.5). The acoustic representation and semantic representation in the internal loop are called predicted acoustics and predicted semantics to distinguish them from the representations resulting from the actual synthesis of the VTL. In the internal loop the errors are calculated between the predicted semantics and target semantics as well as between the predicted acoustics and target acoustics.

The predictive forward model is not a perfect approximation of the VTL but serves as a crude but good enough approximation of the cp-trajectories to acoustics mapping. The training samples for the predictive forward model are created by the VTL and the VTL therefore defines the gold standard and the perfect mapping. The predictive forward model is a good approximation of the VTL for well-known training samples. As the predictive forward model has no insight into the articulation and physics involved in speaking, it can only properly predict the acoustics in regions where the predictive forward model accumulated experience. The prediction-errors are small and informative in regions that are well known to PAULE, whereas rare or completely new articulations are difficult to improve on as the error is large and potentially uninformative. This accumulation of experience is especially desirable for models of language production because of the strong class imbalance of a few high-frequency words, which the predictive forward model should be able to predict in all its variants and many low-frequency words, where the predictive forward model results in predictions and prediction error that is less informative. This experience is not on the word token as a whole, but on time slices of a time resolution of approximately 5 Milliseconds in the cp-trajectories and of approximately 10 Milliseconds in the acoustic representation.

Another advantage of the predictive forward model is that by allowing for an internal loop in an iterative planning process, the acoustic representation and the semantic representation can be internally imagined by the model and no synthesis with the VTL is needed during planning. This imagined acoustic and semantic representation is an expectation on what a given set of cp-trajectories would sound like, if the VTL is used to generate audio from it, and which meaning is associated with this audio. This way of predicting the near future with a predictive forward model is also computationally efficient: the prediction shortcuts the synthesis of an audio wave file and, therefore, predicts in a lower dimensional space which is a lot less oscillatory.

A third advantage comes from the fact that the predictive forward model as well as the embedder, can both implemented with a gradient-aware implementation. Therefore, an error can be backpropagated through these machine-learning models. During the training of these models, this backpropagation is used in learning and updating the internal memory or weights of the models. During planning and adaptation, these gradients can be used to inform PAULE which parts of the cp-trajectories it has to adjust by which amount to achieve a smaller error. Figure 5.7 shows the complete internal loop used by PAULE during planning. The error that comes from the RMSE loss between the

target acoustics and the predicted acoustics and the error that comes from the RMSE between the target semantics and the predicted semantics is complemented by an error on the cp-trajectories themselves. This error on the cp-trajectories is a velocity and jerk loss, which forces the cp-trajectories to be as stationary as possible and if they change that they change with a mostly constant force or acceleration. The velocity for each trajectory of the cp-trajectories is the change in position over time and the jerk is the change in acceleration over time, respectively the first and third derivative in time. A small or zero jerk corresponds to a slowly changing or possibly constant acceleration, which with the proportional correspondence between the acceleration and the applied force ($F = ma$) means a slow changing or possibly constant applied force on the articulators.

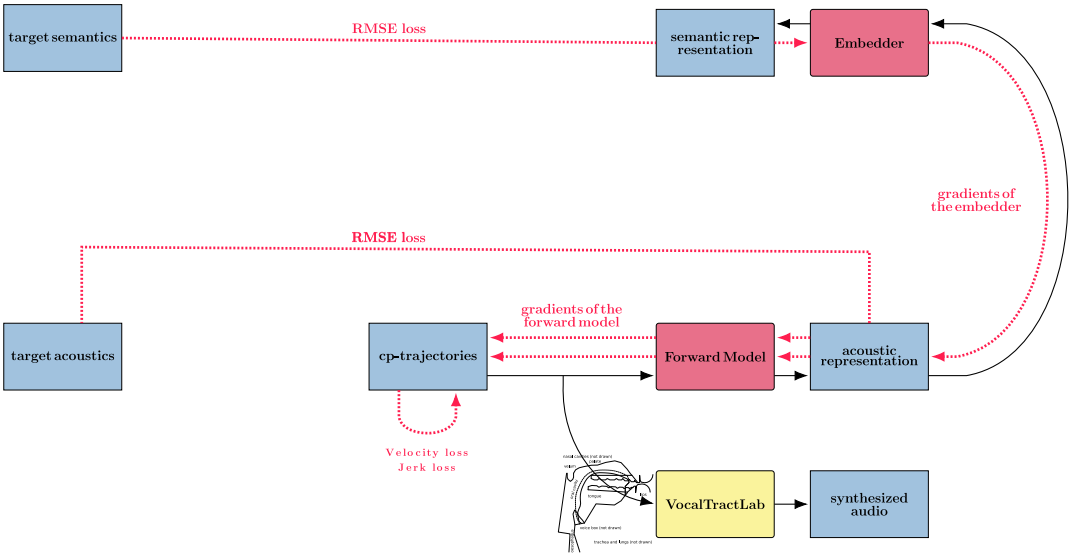

Figure 5.7: Step 6 in the journey to PAULE. By utilising the gradients of the embedder and predictive forward model and adding a low-effort regularization error, a solution for the cp-trajectories for a given target acoustics and target semantics can be derived. This is the internal loop of PAULE, which relies on good information in the gradients of the predictive forward model and the embedder. The low-effort regularizing error is implemented by minimizing velocity and jerk of the cp-trajectories, which fosters stationary trajectories that are changed (if needed) by a constant force. The planning of the internal loop works well if the cp-trajectories are initialized close to a solution of the cp-trajectories. In this figure, this initialization is still missing.

Compared to Figure 5.6, the connecting arrows between the target acoustics and the target semantics to the cp-trajectories were removed in Figure 5.7. This is the possible case since we can close the internal loop by first going through the predictive forward model and the embedder along the black arrows. Now, we compare the predicted acoustics and the predicted semantics with the target acoustics and target semantics. This error is now used and connected back to the cp-trajectories and by that closing the loop through the gradients of the embedder and the predictive forward model. Therefore, no

direct connection between the target acoustics and the target acoustics cp-trajectories is needed anymore. The use of the error and the gradients to close the loop comes with one caveat though. In the very first iteration, we do not only need to initialize the target semantics and the target acoustics but we do need to initialize the cp-trajectories as well. Furthermore, as the gradients of the predictive forward model are only informative for cp-trajectories, which are near cp-trajectories that are already integrated into the experience of the predictive forward model, it is crucial for PAULE to initialize close to the final solution. If the initialization is close to the target semantics and/or to the target acoustics the internal planning loop is extremely flexible and good in adapting the cp-trajectories into a low velocity and low jerk regime while still improving the error in the acoustics and semantics domain. In this way the gaol of optimizing the articulation on the conveyed meaning is achieved.

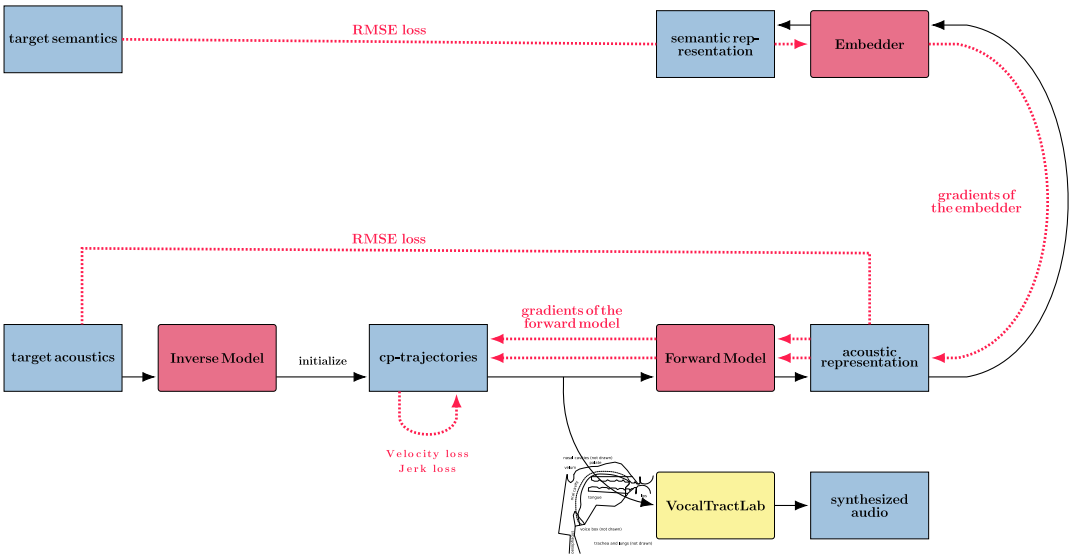

Figure 5.8: Step 7 in the journey to PAULE. In the last step before the final (current) model, a first initialization method is introduced. Namely, the direct inverse model that initializes the cp-trajectories from the target acoustics. Even if the direct inverse model directly maps from the target acoustics to the cp-trajectories and, thereby, seems to solve the original question, it fundamentally cannot find a locally optimal solution by itself. The mapping of the direct inverse model is a one-to-one mapping and cannot condition on the current and past state of the VTL. In reality, that mapping between the target acoustics and the cp-trajectories is a one-to-many mapping. This mapping is achieved in PAULE by planning along the gradients of the internal loop.

In order to solve the initialization of the cp-trajectories, we first created and trained a direct inverse model that produces one set of cp-trajectories for a given target acoustics given as a log-mel-spectrogram (see Figure 5.8). This direct inverse model will always output the same cp-trajectories for a given log-mel-spectrogram as input and, therefore, does not solve the one-to-many mapping. After the initialization with the direct inverse

model the one-to-many mapping is solved by iteratively correcting and planning the cp-trajectories along the three error components of the cp-trajectories themselves, the acoustics, and the semantics.

The direct inverse model only implements initialization from the target acoustics. But because speech production is driven by meaning and meaning should be the starting point, we introduced two GAN models as components to the PAULE model, one which initialize the cp-trajectories from the target semantics and one which create a target acoustics from the target semantics. The full PAULE model is shown in Figure 5.9.

5.2 Planning

Now that we have an understanding of all the boxes and arrows in the PAULE framework depicted in Figure 5.9, in this section, we discuss in detail how the planning works to find suitable cp-trajectories (cf. [21]) and how the planning finds different solutions in different contexts for the same target acoustics and the same target semantics, and with that solves the one-to-many problem. After introducing the planning in PAULE, we go into detail on the implementation of the individual models. For now, the only important properties of the models are, that they are learned and, therefore, incorporate experience and that they can push back an error along the gradients of the model's computations. Additionally, note that the GAN models sample from a distribution and with that can create different cp-trajectories and log-mel-spectrogram from the same semantic vector. This solves the one-to-many problem as well but is not as adaptive and transparent as the planning loop that is the topic of this section.

The planning in the current state of PAULE involves two iterative processes that are interleaved (see Algorithm 1). The first planning is on the internal loop and involves the predictive forward model and its gradients. The second process incorporates further knowledge in the predictive forward model by listening to itself and some potentially good cp-trajectories. This fostering of the knowledge of the predictive forward model is done along the outer loop. The outer loop involves the synthesis of an audio with the VTL. From this audio the produced acoustics and produced semantics is derived. This outer loop is not shown in Figure 5.9, but can be created by connecting the synthesized audio to the acoustic representation and from there going to the semantic representation with the embedder.

The planning itself is the same for all three tasks that PAULE can solve, but the initialization is task-dependent. As the planning can only start after initialization filled all necessary data structures with data, we first explain how initialization is accomplished for each of the three tasks. Specifically, the target semantics, the target acoustics, and the cp-trajectories need to be initialized before the planning can start.

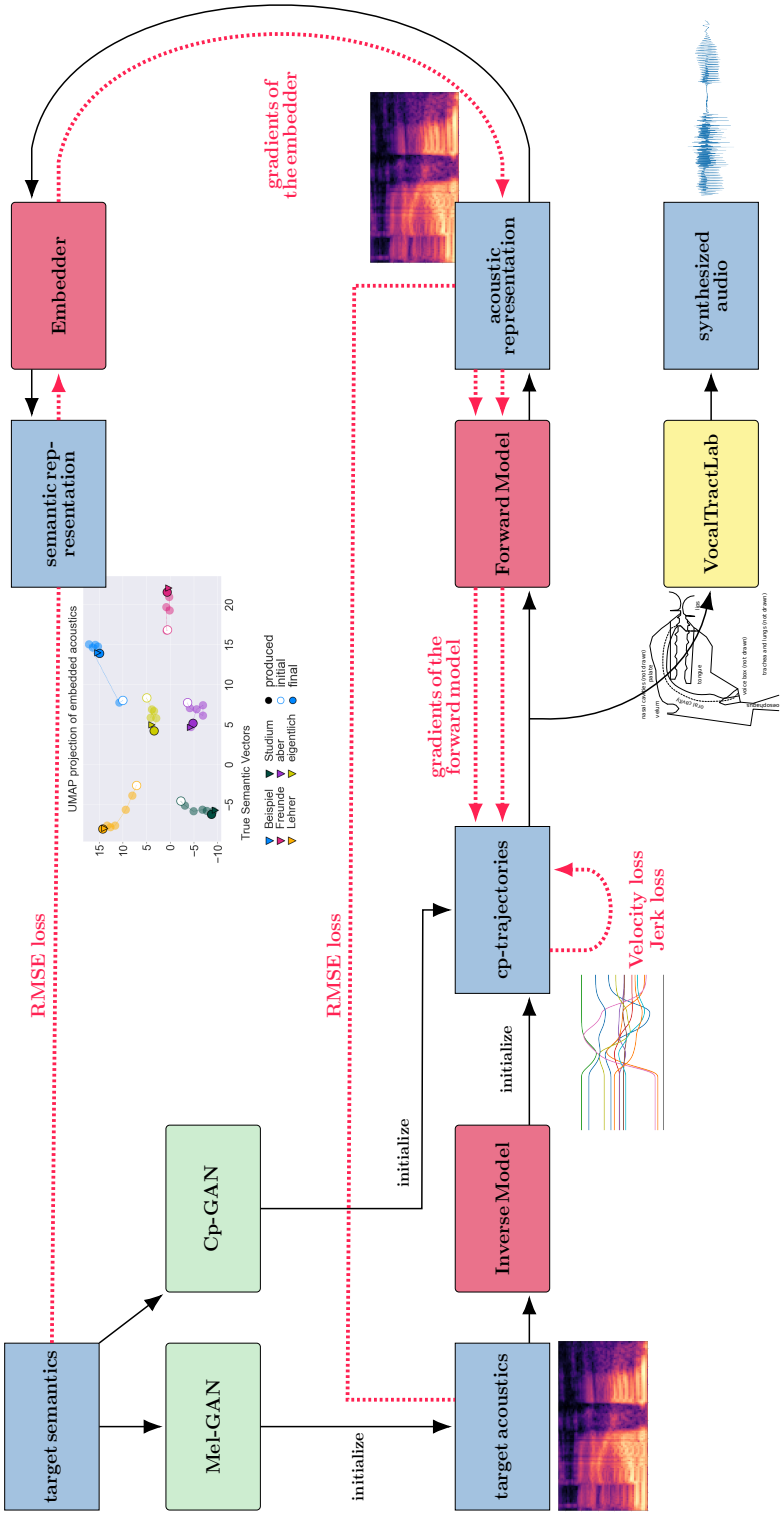

Figure 5.9: The (currently) complete Predictive Articulatory speech synthesis Utilizing Lexical Embeddings (PAULE) framework, which can initialize from a target semantics (semantic-only task), from a target semantics and a target acoustics (semantic-acoustic task), and from the target acoustics (acoustic-only task / copy-synthesis task). Initialization is done by a direct inverse model and/or GAN models. This is followed up by a planning procedure along the gradients of the internal loop. After the planning is complete, the finally planned cp-trajectories are used to synthesize audio with the VocalTractLab (VTL).

Algorithm 1 Algorithm for planning cp-trajectories with PAULE for the semantic-only task. The algorithm uses the following abbreviations. For the data structures it uses: *vec*: semantic vector, *duration*: duration of the final speech sound as integer in steps of 220 audio samples (5 Milliseconds); *cps*: cp-trajectories; *mel*: log-mel-spectrogram. The remaining symbols are: m: model; L: for loss or error; ∇: gradient; α: learning rate; VTL: articulatory synthesizer VTL; $\widehat{mel}$ is the predicted acoustics; $\widetilde{mel}$ is the produced acoustics and *mel* is the target acoustics. Note that we are using stochastic gradient decent in this description but implemented the planning update with the ADAptive Moment estimation (Adam) optimizer[55].

Require: $vec, duration$
  $\widehat{cps} = GAN_{cp}(vec, duration)$
  $mel = GAN_{mel}(vec, duration)$
  for $i = 1$ to 5 (outer loop) **do**
    for $j = 1$ to 24 (internal loop) **do**
      $\widehat{mel} = m_{predictive}(\widehat{cps})$
      $\widehat{vec} = m_{embedder}(\widehat{mel})$
      $L_{planning} = L_{acoustics}(\widehat{mel}) + L_{semantics}(\widehat{vec}) + L_{cps}(\widehat{cps})$
      $\widehat{cps} = \widehat{cps} - \alpha \cdot \nabla L_{planning}$
    end for
    $\widetilde{mel} = VTL(cps)$ for 8 to 24 *cps* from the internal loop
    continue training $m_{predictive}$ with produced $\widetilde{mels}$
  end for
  return $\widehat{cps}$,$\widehat{mel}$ $\widetilde{mel}$

5.2.1 Initialization

The semantic-only task (also called full-generation task) starts with a target semantics and a duration, which defines the duration of the final word. In this task, the target acoustics and the initial cp-trajectories are missing. The target acoustics is initialized with the Wasserstein-GAN trained to sample log-mel-spectrograms (mel-GAN), which takes the target semantics and the duration as well as a random vector as input. In the same way, the cp-GAN initializes the initial cp-trajectories from the target semantics in the semantic-only task.

In the acoustic-only task (also called copy-synthesis task), only the target acoustics is given. The task is to find cp-trajectories that match the target acoustics as closely as possible and in order to fully initialize PAULE, the target semantics, as well as the initial cp-trajectories, need to be computed. The initial cp-trajectories are created from a direct inverse model, which takes the target acoustics as input. The target semantics is computed from the target acoustics through the embedder.

In the semantic-acoustic task both the target semantics and the target acoustics are

given by the end user, but the initial cp-trajectories are missing. These can now be either initialized from the target semantics or from the target acoustics. In the current implementation, the direct inverse model gives better results compared to the cp-GAN and, therefore, the default is to use the direct inverse model in the semantic-acoustic task for now.

5.2.2 Internal loop

The internal loop in PAULE is visualized in Figure 5.7 and is the most inner for-loop in Algorithm 1. The task of the internal loop is to iteratively correct the cp-trajectories along a predicted acoustics, a predicted semantics, and a cp error.

A single execution of the internal loop starts with the prediction of the log-mel-spectrogram of the predicted acoustics from the current cp-trajectories by the predictive forward model. From this predicted acoustics, a predicted semantics is derived with the embedder. Now, the predicted acoustics is compared to the target acoustics, the predicted semantics is compared to the target semantics and for both the RMSE error is computed. Additionally, the RMSE is computed between the velocity (first-time derivative) of the cp-trajectories and flat zero-line trajectories (no velocity). In the same way, the RMSE of the jerk (third-time derivative) of the cp-trajectories and zero-line trajectories (no jerk) are computed.

The four error components (acoustic, semantic, velocity, jerk) are added together to form an additive loss. This combined error is then pushed back along the gradients of the embedder and predictive forward model for the semantic error; along the gradients of the predictive forward model for the acoustic error; along the velocity computation for the velocity error; and along the jerk computation for the jerk error.

The pushed-back error informs us in a linear approximation of how to change the current cp-trajectories to minimize the additive error. To calculate new cp-trajectories from the current cp-trajectories and the pushed-back error Stochastic Gradient Decent (SGD) could be used as the simplest update scheme, but we applied the ADAptive Moment estimation (Adam) optimizer[55] during the planning in the internal loop. Adam not only takes the pushed-back errors from one iteration into account, but additionally takes the information of the last planning updates into account as well. The Adam optimizer is initialized with a relatively large learning rate of 0.01, but can be set manually to other values when calling the planning in the `paule` python package.

After the pushed-back error is applied to the current cp-trajectories, the newly planned cp-trajectories become the current cp-trajectories and the internal loop is started over again. We default to 24 repetitions for the internal loop, but depending on the quality of the initialization and the accumulation of the predictive forward model, more iterations might give better results. We tested up to 100 internal iterations, which resulted in slightly improved results, but were not worth the huge additional computational costs.

5.2.3 Outer loop

In the outer loop, the PAULE model synthesizes audio with the VTL from cp-trajectories and listens to itself. From the synthesized audio the produced acoustics and the produced semantics is computed. This results in triplets of data points, a cp-trajectories with the associated acoustic and semantic representation. The main purpose of the outer loop in the PAULE model is to improve the predictive forward model by exposing it to the cp-trajectories, produced acoustics pairs in supervised learning. The outer loop is necessary whenever the predictive forward model does not give informative gradients on how to improve the current cp-trajectories. In the outer loop, no planning on the cp-trajectories is done, but the experience of the predictive forward model on the articulatory model is improved in the region of the planning at hand. This use of the outer loop can be seen as the model articulating the word it has in mind to itself, listening to it, and then improving the articulation further after remembering this practising articulation.

The integration of the new experiences into the predictive forward model is implemented by further training the weights of the predictive forward model with new training samples that are generated from planned cp-trajectories from the internal loop and the corresponding synthesis results from the VTL. We will call this process of adding new experiences during the planning *practicing*.

The new samples for practicing are created by synthesizing 8 to 24 cp-trajectories from the internal loop with the VTL and calculating the log-mel-spectrogram for the synthesized mono audio. The learning is continued with an Adam optimizer, which is different from the one used in the internal loop and is initialized with a substantially smaller learning rate of 0.001. The training samples are exposed to the predictive forward model 10 times, i. e. we default on 10 epochs in the practicing training.

For some time, we seemed to experience catastrophic forgetting in the predictive forward model as the training samples in the practicing training of the outer loop are very similar and only training on these makes the predictive forward model memorize these and forget all the other accumulated knowledge. This catastrophic forgetting resulted in completely uninformative and very high gradients. To mitigate the catastrophic forgetting, we mixed some old training samples into the practicing training data. But in the end, this was not necessary anymore. Still, the option to add training samples for the practicing training is still in the code and optionally available.

At the moment, we only implement practicing for the predictive forward model. In principle, it is possible to implement it for the GAN models as well as for the direct inverse model. This is a good idea, but increases computation costs without getting any direct benefit for the single planning at hand, as these models are only used during initialization. Furthermore, this makes the planning in PAULE even less reproducible and deterministic, which scientifically speaking is not desired.

5.3 Theory

Before we discuss the different internal models used in PAULE, two important concepts are introduced here. The first concept is the local gradient, which allows for error back-propagation and informs the iterative planning process in PAULE. Why the automatic calculations of local gradient information is important is discussed as the first topic of this section. The second topic is the Long-Short-Term-Memory (LSTM) cell and a LSTM-layer in an Artificial Neural Network (ANN) model. As we have time-series data with a sequence length of 50 to 400 time-steps and as the data is highly autocorrelated, i. e. follows relatively smooth curves in time, it is important to model these autocorrelative dependencies. LSTM-layers are designed to properly deal with variable length autocorrelated sequence data.

Other techniques like drop-out, adding random noise to the learning data, residual nets, and rectifying linear units (ReLUs) have been explored for the PAULE model but are neither necessary nor helpful in understanding the concepts behind PAULE. As we aimed for simplicity, these techniques are gone from the important individual models presented in Section 5.4 and will not be discussed in detail here.

5.3.1 Autograd & gradient-aware models

The first important development of the last years, which allows models like PAULE to do their trickery, is the widespread availability and easy use of gradient aware calculation pipelines. These pipelines are based on computational graphs and use a generalised form of a vector or matrix, called tensor. An input tensor is transformed into an output tensor using mathematical operations (the computational graph). Given the output tensor, an error signal is calculated based on a predefined objective function, also called *loss* function in machine learning. The calculated error or loss can be pushed-back through the computational graph and is used to update the free parameters of the mathematical operations (weights and biases) or, in case of the PAULE model, update the input parameters. The only requirement for the mathematical operations is that the first derivative, the gradient, can be approximated, and in the best case is analytically given. The approximated gradient does not have to be the exact mathematical gradient, but should be close enough to get the models to work properly.

Two famous frameworks that support gradient aware computational graphs that can be interfaced from the Python programming language are *PyTorch*[70] and *Tensorflow*[27]. In this thesis and for the `paule` python package, we used PyTorch in version 1.8.1. All models are implemented by creating computational graphs. The inputs are defined as tensors that consist of a batch dimension, a channel dimension (cp or log-mel-spectrogram), and a time-step dimension or sequence dimension. The batch dimension has a fixed size, the so-called batch size, that is set to 8 (in some cases to 64) in our work. The channel size is fixed to 30 for the cp, to 60 for the log-mel-spectrogram, and to 300 for the semantic vector tensors. The last dimension has a variable size of around 50 to 400 time-steps and is not present for the semantic vector tensor.

In the forward pass, an input tensor is given to the model and the model's computational graph is applied to the input tensor consecutively. Within the computational graph entries of the input tensor are changed by multiplying weights and adding biases before applying a usually non-linear activation function. The multiplication with the weights and the addition of the biases is done according to the tensor product, which is a generalized matrix product and is itself a linear operation. The non-linearities only come from the explicit non-linearities that are applied to the activations after the tensor product is computed. During the forward pass, the model stores the intermediated activations as far as they are needed to compute the gradients in the backward pass. Tensor shapes, i. e. the number of dimensions and the size of each dimension of the tensor, can change during the computations from the computational graph.

After all computations of the computational graph are computed in the forward pass, the output tensor is generated. This output tensor can now be compared to a reference or target tensor of the same shape. The differences between the output tensor and the reference tensor can be computed according to different distance metrics, which are defined by the loss function or learning objective. In this thesis the objective or loss functions consist mostly of the RMSE. The L1-norm and the L2-norm were explored as well, but did not yield similarly good results. Given a specific distance metric or error measure, the loss can be calculated. This loss (or error) is calculated for each cell in the output tensor, and is the starting point of the backward pass.

In the backward pass, the gradients computed by autograd show their power. After calculating the loss for all cells in the output tensor, the computed gradients are used to distribute the error backwards from the output tensor to the intermediate layer directly before the output tensor in a first step. In the next step, the distributed loss can be used to calculate the error for the intermediate layer before that one. These step-by-step computations in reverse order, therefore the name backward pass, calculate the contribution of each activation from the forward pass to the final error. To make things computational efficient for the framework first order derivatives for all non-linearities are required to be analytically given while relevant information is stored during the forward pass.

At the end of the backward pass, each connection weight or bias involved in the forward pass now has a gradient attached to it, which quantifies its contribution to the final error and at the same time specifies what change to the number is needed to minimize the final error under a linear approximation. Although this linearity assumption is generally not fulfilled, the Taylor-expansion tells us that the linear approximation is arbitrarily good for a (very) small variation of the output tensor and the activations. This Taylor-expansion exists and has this property as all the computational steps are differentiable. Therefore, changing the activations only a little is guaranteed to make the final error smaller. Unfortunately, it is unknown how small this step should be. The size of the step is called the learning rate and is a so-called hyper parameter, which must be manually set. While small learning rates guarantee to reduce the final error, the reduction is small. In contrast, large learning rates potentially reduce the final error substantially, but might in the worst case increase the final error because the linearity

approximation might be strongly violated.

Overall, the combination of linear algebra on tensors, explicitly defined non-linearities with corresponding derivatives, as well as the automatic bookkeeping and computations of forward and backward pass allow for the supervised learning of big machine learning models. PAULE exploits all of these mechanisms to backpropagate an error signal to the inputs of an articulatory speech synthesis system, namely the cp-trajectories of the VTL. This process of updating these inputs based on the error signal is called planning.

5.3.2 Long-Short-Term-Memory (LSTM)

With the decision not to use any symbolic representation on the acoustic or motor level, articulatory speech synthesis in PAULE is modeled in a time-series to time-series and time-series to fixed-vector manner. This decision leads to the problem of dealing with long-ranging dependencies in highly correlated, multi-channel time-series data. A powerful building block to handle this problem is the Long-Short-Term-Memory (LSTM)-cell and the ANN-layer that can be created with these cells.

Modeling time-series or sequences naively with ANN architectures lead to the development of Recurrent artificial Neural Network (RNN) architectures. RNNs work by recursively feeding the model output activations of the last time-step as inputs into the model's next input activations. This is usually done together with new input data for the next time-step. Therefore, for each time-step, the inputs and outputs are connected by the same model and the correlative structure can be modeled by aggregating the relevant information of the past into the activations that are passed in as new inputs to the model at the next time-step. The same mechanism is applied within a LSTM-network, which enables the model to deal with arbitrary long time sequences with the same number of weights. The number of time-steps only changes the number of forward passes through the model.

A common way to train RNN and LSTM models calculates the gradients not only by layer, but also per time-step. This method is the so-called *backpropagation through time* method. During the forward pass, at each time-step the same RNN or LSTM-model is called once. Activations of the full RNN-model for each individual time-step are stored separately. In the backward pass, not only the model is traversed backward, but additionally, the time sequence is traversed backward. Gradients of the model are calculated once per time-step. With a time sequence length of 400 time-steps, the gradients are calculated and stored 400 times compared to the single model. For an update of the weights and biases all 400 errors are summed together for each cell. This backpropagation through time method results in the following severe problem for RNN models, which is mitigated by the LSTM-network architecture.

The analytic calculations of the gradients involve the repeated use of the chain rule of differentiation, which leads to a multiplicative relationship between terms in the gradient computation. These terms are all very similar and usually evaluate to a similar number. The number of terms depends on the depth of the layer and on the position in the time-sequence in backpropagation through time. At early time points more terms

accumulate during the backpropagation steps. With a multiplicative set-up, this leads to gradients that are either virtually zero – known as the vanishing gradient problem – or that are extremely (!) large in their absolute value – known as the exploding gradient problem. For example, for a very simple model with 400 time-steps, the gradient of the first time-step consists of 400 terms all connected by multiplication. The resulting gradient therefore can be approximated by the average evaluation of all terms raised to the power of 400. If each term evaluates to around 0.9 the gradient is practically zero with $0.9^{400} \approx 5.0 \times 10^{-19}$. If all terms evaluate to around 1.1 the gradient would be around $1.1^{400} \approx 3.6 \times 10^{16}$.

The vanishing and exploding gradient problem result in learning regimes for RNN models that require a very small learning rate to ensure that the gradients don't explode. Unfortunately, this small learning rate brings the backpropagation through time into the vanishing gradient regime, where the informative error can only be pushed back in time over very few time-steps. Therefore, only very short-ranging autocorrelations can be modeled with RNNs.

This problem could be substantially alleviated by the introduction of LSTM-cells[47]. As a special kind of RNN, LSTM-models implement a so-called *error carousel*. Figure 5.10 shows the internal structure of one LSTM-cell. If the output gate is closed, i. e. has an activation near zero, the multiplicative gradient terms are canceled out and the main contribution of the gradient is directly transferred from the cell state one time-step in the future. In the forward pass, the cell state is written by the input gate and read out by the output gate. Furthermore, the LSTM-cell contains a forget gate that allows to erase the cell state.

All the non-linear activation functions are sigmoid functions (σ) with output values between 0 and 1 and the tangens hyperbolicus (*tanh*) with output values between -1 and 1. The derivatives (gradients) of these activation functions are smooth differentiable, mono-modal functions without any jumps, which give a meaningful gradient in the range of -1 to 1. Smooth derivatives of the activation functions leed to nicely behaving gradients in the backward pass and during planning of PAULE. As the activation functions of the LSTM are most sensitive in the -1 to 1 range, a normalization of the training data into that range is crucial. In principle, data not in the -1 to 1 range could be modeled with LSTM-cells as well, when the weight initialization is done according to the data ranges, but often it is more transparent and easier to map all the data into the -1 to 1 range. Note, however, that it is not advisable to use any non-linear data transformations for rescaling the data as this changes the distance structure in the data. There might be good reasons for non-linear scaling of data, but this non-linear scaling should not be motivated with the use of LSTM-cells. Although LSTM-cells are capable to handle data outside of the -1 to 1 range, substantially more training time would be needed due to the limited sensitivity outside of this range.

Each LSTM-cell contributes four parameters, which must be trained. Stacking many LSTM-cells together forms a LSTM-layer, which itself can be further stacked or connected to different layers by dense connections. Due to the recurrent design of the LSTM-cells, activations of the next time-step depend on the prior-time-step during the forward

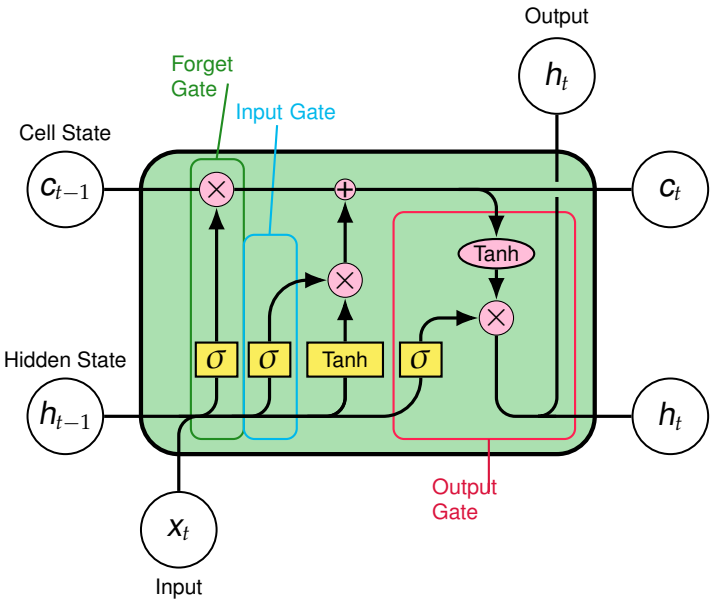

Figure 5.10: The Long-Short-Term-Memory (LSTM)-cell. The multiplicative connections between the hidden state h_{t-1} and different expressions of the cell state c_{t-1} lead to different gates that allow for an error carousel in the backpropagation step and effectively mitigates the exploding and vanishing gradient problem prevalent in Recurrent artificial Neural Network (RNN) architectures. The LSTM is a specific form of RNN that explicitly addresses this problems with its gating mechanisms.

pass. This is the case for all RNN models. Even though this temporal dependency is theoretically appealing, both RNN and LSTM have a computational disadvantage, namely, that calculations cannot be parallelized. As a consequence, these models cannot take full advantage of the computational power of current graphics cards.[1]

With the error carousel in place and the smart gating of the LSTM-cell, it is possible to model long-ranging relationships in time-series to time-series data and in time-series to fixed-vector data. The RNN nature of LSTM-cells makes it possible to model variable length sequences, where the sequence length is only limited by the available computational memory resources. Except for the GAN networks, LSTM-layers are present and are the work-horse in all internal models of PAULE. The GAN models are not implemented using LSTM-layers as they were extremely slow to train. Implementations with LSTM models for GANs exist, and preliminary training showed promising results, however, in order to keep computation times within reasonable bounds and to avoid large environmental footprint for PAULE, we did not pursue using LSTM-cells for GANs.

5.4 Individual component models

PAULE consists of several individual component models that connect the different data structures (see Figure 5.9). The models are Artificial Neural Network (ANN) models trained with data from the Mozilla Common Voice corpus (Common Voice) or the IMS GECO database (GECO) (see Chapter 4). Except for the GAN models, all individual models are trained in a supervised learning fashion with an RMSE loss function and an Adam optimizer. All models, except for the GAN models, make heavy use of at least one LSTM-layer.

Training is performed using batches of typically 8 samples. All samples in one batch are shown to the model in parallel. For each training sample in the batch, a forward pass, a loss computation, and a backward pass is conducted. As all training samples are shown in parallel, independent learning between the samples is assumed. Computation time is practically identical for the training on a batch compared to training on single training samples. Additionally, training with batches is more robust and statistically stable as the errors are summed over all samples in the batch for each weight. This leads to an averaging effect, which effectively allows the error to be minimized for all samples jointly. For too large batches, this averaging effect might, in the worst case, erase all relevant information, especially if the models are highly non-linear, i. e. the error landscape is very wiggly.

As we deal with time-series data that is different in its duration or sequence length, batches usually include samples that are very different in length. In order to compute

[1] Transformers [115] are specifically designed to model sequence to sequence data like time-series to time-series data while being parallelizable and do not rely on a recursive or recurrent pattern and are not RNN even if they solve similar tasks. Therefore, with respect to future progress, it might be interesting to switch to a Transformer architecture from the LSTM-models used for PAULE right now. LSTM-layers used in PAULE follow a vanilla flavour, namely unidirectional forward LSTM-layers trained with backpropagation through time.

such batches, the shorter training samples need to be padded to all conform to the longest sequence in the batch. The padding should be done in such a way that the padded values do not contribute to the error during training and still be computationally efficient. While one reason to apply batch learning is to decrease the computational costs during training, the necessary padding can harm the training process. Benefits of the batch training diminish at the end of the batched time-series, where only the time points of one training sample are left.

As we had problems getting proper padding to work and the improper padding we used initially resulted in either extreme slow computation times or artifacts at the end of the time-series, we adopted same-size batching in the end. In same-size batching, the batches are not created by randomly selecting training samples from the whole training data set, but by first ordering the training samples by the length of their time-series. Subsequently, batches are formed out of time-series training samples of the same or very similar length and padding is kept to a minimum. The application of same-size batching removed all artifacts we encountered beforehand at the end of the time-series in the model predictions.

The following subsection introduces and discusses the individual models. Their role within PAULE is described, a precise model definition is given, and the (supervised) training of the model is described and evaluated. We start with the predictive forward model and the embedder, which both play a central role in the internal loop in PAULE and allow for the predictive nature of PAULE. Details on the implementation are available in the `paule` package in the `models.py` Python module.

5.4.1 Predictive forward model

The predictive forward model is the key component in PAULE. It maps the cp-trajectories directly onto a log-mel-spectrogram, i. e., the motor commands onto the acoustic representation. The predictive forward model 'imagines' the acoustics resulting from the upcoming articulation. These internally imagined predicted acoustics can then be compared to the target acoustics. The resulting error between the prediction and the target is then used together with the gradients of the predictive forward model to plan and further improve the cp-trajectories.

In addition to the predictive nature that the predictive forward model implements, the predictive forward model accumulates experience on the articulatory speech synthesizer VTL and, therefore, allows PAULE to be intrinsically dependant on the experience it has accumulated over learning with the language. For a cognitive model of speech production, this is an essential design feature as different speakers with different speaking background and different vocal tracts articulate the same linguistic word types systematically different.

Architecture

Different from the rest of the models presented later, we will discuss two versions of the predictive forward model. We do this to show how simplification took place during

the development of PAULE and to make the reader aware that each of these models could be exchanged for another model, as long as the main rationale behind PAULE stays intact, namely the mapping of the respective inputs to the corresponding outputs and the capability of pushing back an error. Therefore, it is not critical to understanding every detail in individual models, as long as the overall structure and the function of the models are understood.

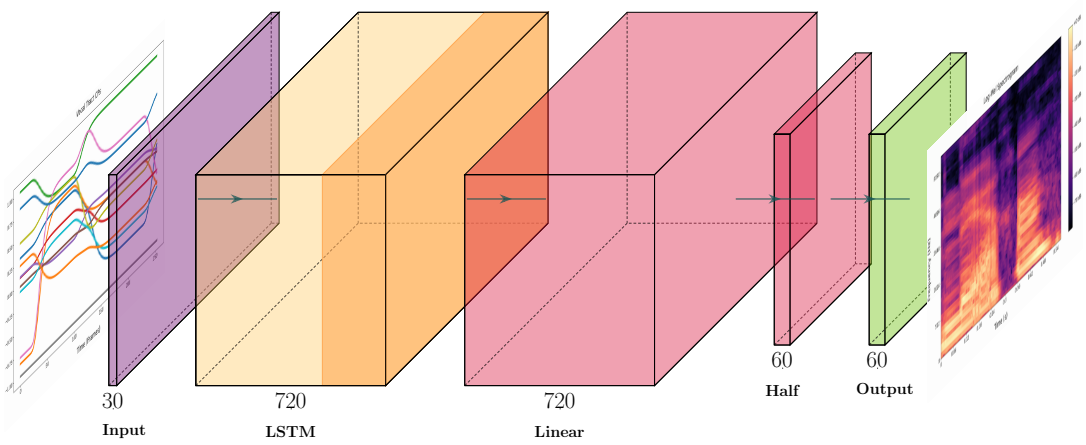

Figure 5.11: The architecture of the predictive forward model. The predictive forward model takes cp-trajectories as inputs and outputs a log-mel-spectrogram, which is used as the acoustic representation throughout PAULE. The architecture consists of a single 720 cell-wide LSTM-layer and some reshaping operations.

We start with the simpler model, which is used in the current version of PAULE (version 0.2.0). The model architecture is depicted in Figure 5.11 and the Pytorch model definition is given in Listing 5.1. The model takes the cp-trajectories as inputs, therefore the input tensor has a shape of *batch size* $\times$ *30* $\times$ *sequence length*. This input is fed into a LSTM-layer of 720 cells mapping to a tensor of shape *batch size* $\times$ *720* $\times$ *sequence length*. Afterwards, this tensor is mapped onto a 60 dimensional tensor using a linear layer. The tensor now has a tensor shape of *batch size* $\times$ *60* $\times$ *sequence length*. This is already the number of channels needed for the log-mel-spectrogram output format, but as we opted for having the time resolution only twice as coarse in the log-mel-spectrogram compared to the cp-trajectories, the sequence length has to be halved. Therefore, two consecutive time points are pooled by a final average pooling layer. The output tensor has a shape of *batch size* $\times$ *60* $\times$ *half of the sequence length*. To not run into problems in the halving step, input cp-trajectories are same-padded to an even sequence length.

Listing 5.1: Pytorch definition of the predictive forward model.

```
1 class PredictiveForwardModel(torch.nn.Module):
2
3     def __init__(self, input_size=30,
```

```
4                 output_size=60,
5                 hidden_size=180,
6                 num_lstm_layers=4):
7         super().__init__()
8
9         self.half_sequence = torch.nn.AvgPool1d(2, stride=2)
10        self.lstm = torch.nn.LSTM(input_size,
11                                  hidden_size,
12                                  num_layers=num_lstm_layers,
13                                  batch_first=True)
14        self.post_linear = torch.nn.Linear(hidden_size, output_size)
15
16    def forward(self, x, *args):
17        output, _ = self.lstm(x)
18        output = self.post_linear(output)
19        output = output.permute(0, 2, 1)
20        output = self.half_sequence(output)
21        output = output.permute(0, 2, 1)
22
23        return output
```

This architecture is substantially simpler compared to the predictive forward model used in the 2020 publications [95,93]. Simplifying the model slightly decreased the accuracy of the validation and test data but gives at least as good gradients during planning and has substantially faster execution times.

The architecture of the model used in 2020 is shown in Figure 5.12. Here, the input cp-trajectories are first modified by five residual convolutional layers in the time dimension. Next, from the enhanced location inputs, velocities and accelerations are computed and concatenated to the locations, which results in a tensor of shape *batch size* $\times$ *90* $\times$ *sequence length*. This tensor serves as input to four LSTM-layers with 180 cells each. After the LSTM-layers, a linear-layer maps the LSTM-activation tensor of shape *batch size* $\times$ *180* $\times$ *sequence length* to the final number of 60 channels and an average pooling halves the number of time-steps. The resulting output tensor is again a tensor of shape *batch size* $\times$ *60* $\times$ *half of the sequence length*.

This old predictive forward model from 2020 was one out of several different ones we tested, which gave the best performance in terms of minimal loss on the held-out test data. Compared to the much simpler model used currently, the old model had similar planning capabilities but was substantially slower to execute. Since the predictive forward model is executed in every iteration of the internal loop, the execution time is critical for the whole planning procedure and, therefore, for the overall speed of speech synthesis by PAULE. Additionally, the hold out test data contains cp-trajectories that are created by the segment-based synthesis approach and therefore do not follow the distribution of the cp-trajectories created through planning with PAULE. Therefore, the performance measured on the held-out test data only roughly approximates the performance of the predictive forward model in its usage within PAULE.

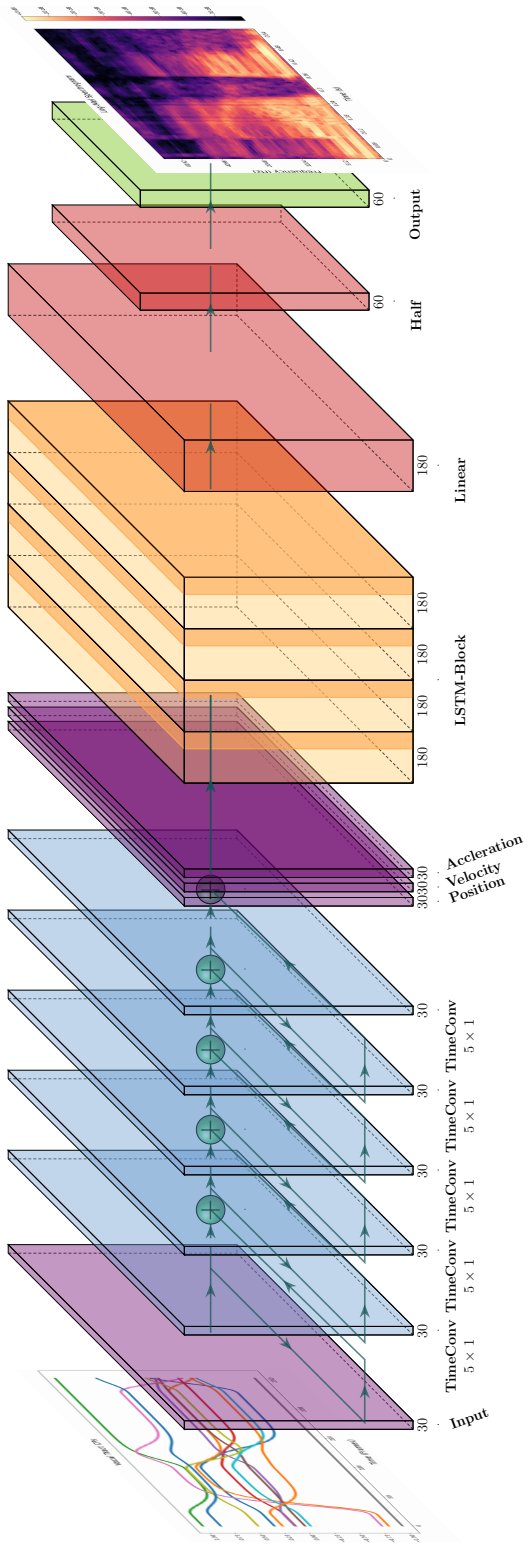

Figure 5.12: The architecture of an outdated predictive forward model used in 2020. The architecture is a lot more complicated compared to the current architecture of the predictive forward model and consists of a stack of residual convolutions followed by a stack of four 180 cell-wide LSTM-layers, followed by some reshaping operations. This model is shown to give an impression on the simplifications we could achieve in the end for the internal models in PAULE without losing substantial amounts of accuracy.

Training & evaluation

The current predictive forward model is trained on the resynthesized segment-based part of the Common Voice data set with 21,175 word tokens and 4,311 word types in a supervised learning fashion. Batch normalization is used with a batch size of 8. The model is trained over 100 epochs, where one epoch is one full pass through the training data set. With 100 epochs each sample is presented 100 times to the model during training. That the predictive forward model can only be trained on cp-trajectories, log-mel-spectrogram pairs that are created with the segment-based synthesis approach (see Section 4.1). On the one hand, the knowledge engineered into the segment-based cp-trajectories may benefit the predictive forward model. On the other hand, the predictive forward model is exposed to rule-based synthesis data that are understandable but sound quite artificial, and the variability across tokens of the same type that is naturally present in real speech is absent. As a consequence, predicting the audio of highly variable audio tokens is a hard task: The input data come from a different statistical distribution than the output target distribution. It is therefore unsurprising that PAULE also relies on the outer loop for learning. (Which, from a cognitive perspective, makes sense: we hear ourselves when we speak.)

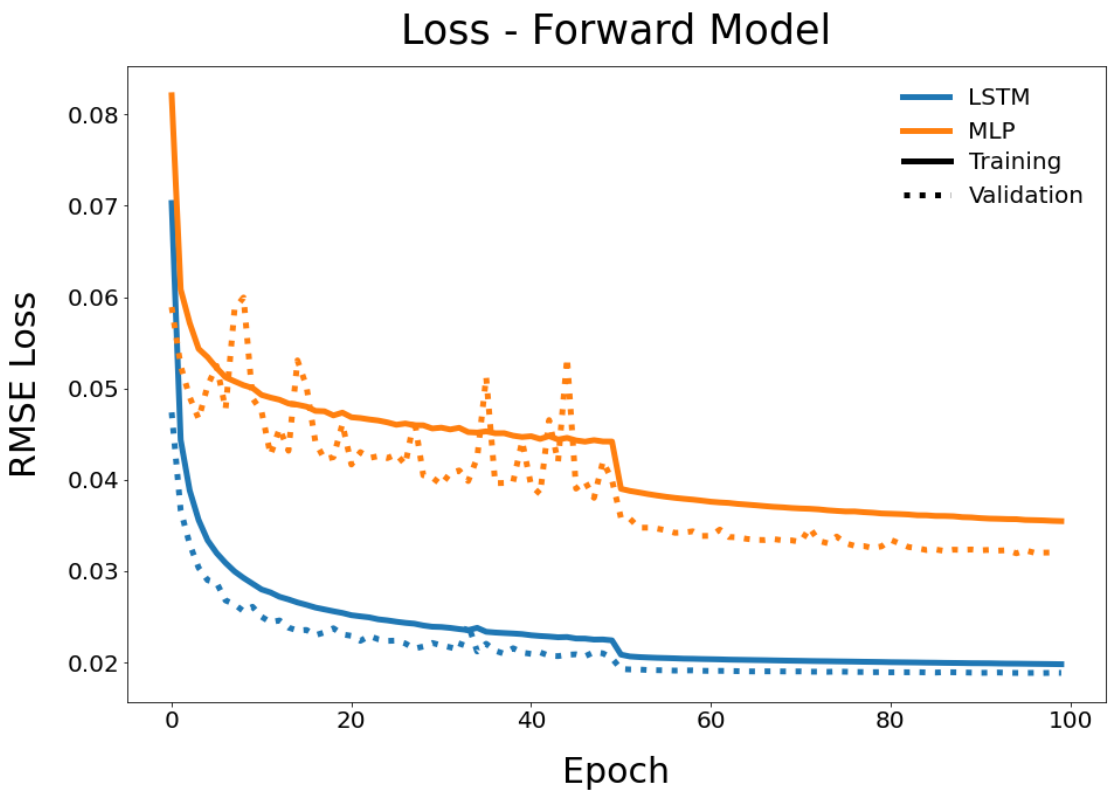

Figure 5.13: The loss of the predictive forward model compared to a Multi-Layer-Perceptron (MLP), which after 100 epochs still has a substantially higher loss.

Figure 5.13 shows that the loss is reduced from an initial value of 0.063 to 0.017, which is substantially smaller compared to a non-recurrent feedforward Multi-Layer-Perceptron (MLP) trained on the same data for comparison. The loss itself might be misleading as it is not informative about weather the log-mel-spectrogram are approximated well at all. Therefore, it is necessary to look at least at some of the predictions of the model and compare these to the ground truth data. Figure 5.14 presents the various log-mel-spectrogram for three tokens of the word /Lehrer/ (teacher). The log-mel-spectrogram of the recordings illustrate the variability in the ways this word is pronounced. For the first two tokens, /Lehrer/ is produced with two syllables. The third token illustrates a single-syllable realization. The predicted log-mel-spectrogram are more similar to the VTL-resynthesized log-mel-spectrogram than to the actual log-mel-spectrogram calculated from the recordings, as expected: the predictive forward model is never exposed to the log-mel-spectrogram of the actual recordings. Clearly, both the LSTM-based predictions and the MLP-based predictions well-approximate the gold-standard log-mel-spectrogram that they are trained to predict.

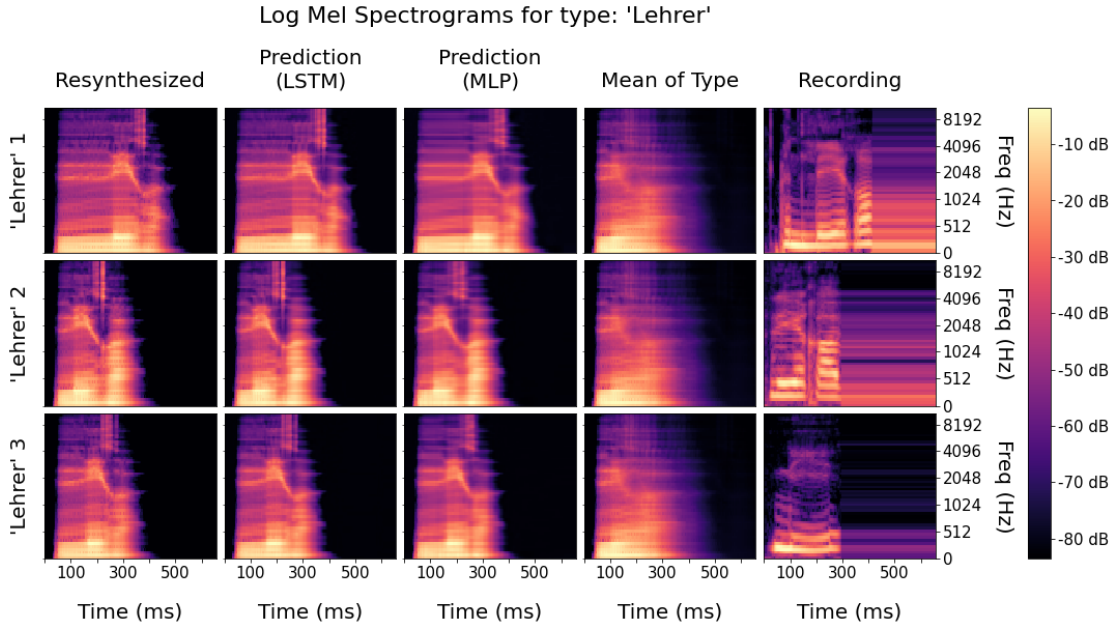

Figure 5.14: Three tokens of the word /Lehrer/ (teacher) from the GECO. Example predictions by the predictive forward model and by the Multi-Layer-Perceptron (MLP) are compared to the ground truth (Resynthesized), the mean log-mel-spectrogram over the word type, and associated human recordings.

Both the MLP as well as the LSTM-based predictive forward model achieve to model the structure of the cp-trajectories to log-mel-spectrogram mapping created by the VTL.

5.4.2 Embedder

The embedder is the second key model in PAULE. It maps the log-mel-spectrogram onto a semantic vector, i. e., acoustic representation onto the semantic representation. In contrast to the predictive forward model, the embedder has no physical model like the VTL to approximate. Therefore, the mapping of the embedder is used in two ways: in the outer loop it maps the produced acoustics onto the produced semantics and in the internal loop it maps the imagined predicted acoustics onto the predicted semantics. This twofold usage arises from the fact that there is no available ground truth simulation for the construction of meaning from speech. If we would have a human listener in the outer loop, this human listener could serve as the ground truth (like the VTL does for the predictive forward model). Without a human listener in the loop, the embedder gives both, the imagined predicted semantics as well as the best estimate of the actual produced semantics. In contrast to the predictive forward model, this twofold usage does not allow to incorporate experience in the embedder during the planning of PAULE. Still the embedder is a very important component of PAULE as for the planning in PAULE, the RMSE error between the predicted semantics and the target semantics is used to improve the cp-trajectories. Additionally, the embedder is used in the synthesis of the acoustic-only task to derive an initial target semantics from the log-mel-spectrogram of the target acoustics.

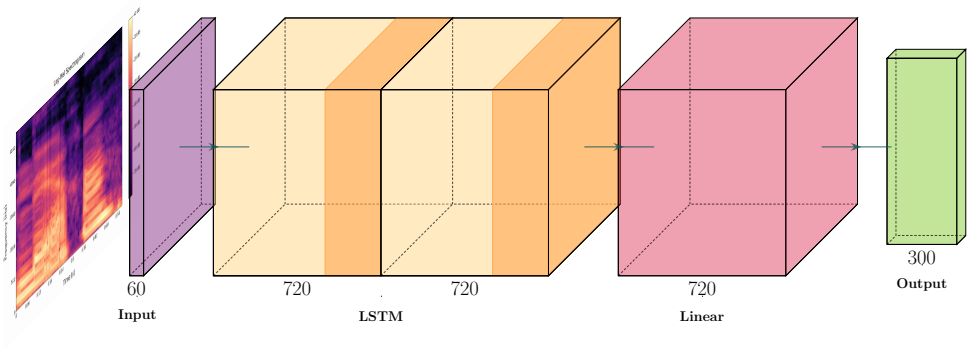

Figure 5.15: The embedder model consists of two LSTM-layers followed by a linear layer at which the last activation is extracted and used as the prediction of the semantic vector. The embedder maps the variable length time-series data of the log-mel-spectrogram onto the fixed length semantic vector representation.

As the embedder is used in every iteration of the internal loop, execution times are critical for the overall planning speed of PAULE. The architecture (see Figure 5.15 and Listing 5.2) reflects this by having again an as simple as possible approach for the time-series to fixed-vector model. The input log-mel-spectrogram with a tensor shape of *batch size* $\times$ *60* $\times$ *sequence length* is followed by two LSTM-layers with 720 cells each. From the output tensor of shape *batch size* $\times$ *720* $\times$ *sequence length* information at the

last meaningful time-step in the input log-mel-spectrogram is extracted resulting in a 720-dimensional vector. For all samples in the batch, these vectors are stacked together to form a tensor of shape *batch size* $\times$ *720*, which then is linearly mapped to the final output shape of *batch size* $\times$ *300*.

Listing 5.2: Pytorch definition of the embedder.

```
class Embedder(torch.nn.Module):

    def __init__(self, input_size=60,
                 output_size=300,
                 hidden_size=720,
                 num_lstm_layers=2,
                 post_activation=torch.nn.LeakyReLU(),
                 dropout=0.7):
        super().__init__()

        self.lstm = torch.nn.LSTM(input_size,
                                  hidden_size,
                                  num_layers=num_lstm_layers,
                                  batch_first=True,
                                  dropout=dropout)
        self.linear_mapping = torch.nn.Linear(hidden_size, output_size)


    def forward(self, x, lens, *args):
        output, (h_n, _) = self.lstm(x)
        output = torch.stack([output[ii, (last - 1).long(), :]
                              for ii, last in enumerate(lens)])
        output = self.linear_mapping(output)

        return output
```

A large number of different embedder architectures were tested, some of which were substantially more complex compared to the one presented here. In the end, the simple two-layer LSTM model turned out to be the best compromise between good accuracy, fast execution times, good gradients for the planning, and simplicity.

Training & evaluation

The training for the embedder is conducted on both the segment-based resynthesized acoustics and the acoustics from the recordings of the Common Voice corpus. This leads to a training data set of 42,350 word tokens and 4,311 word types. In order to enforce a better gradient structure in the embedder, normally distributed i.i.d noise is added to each target semantic vector dimension with a mean of 0 and standard deviation of 1/6 of the minimal distance of the semantic vectors in the training data. New noise is added at each epoch. For the two LSTM-layers, dropout was used and the training ran over 200 epochs. Training time-series to fixed-vector models is more difficult and usually needs

longer training times. The training loss dropped from 0.45 to 0.03 (see Figure 5.16) and is substantially better compared to an MLP trained on the same task for comparison.

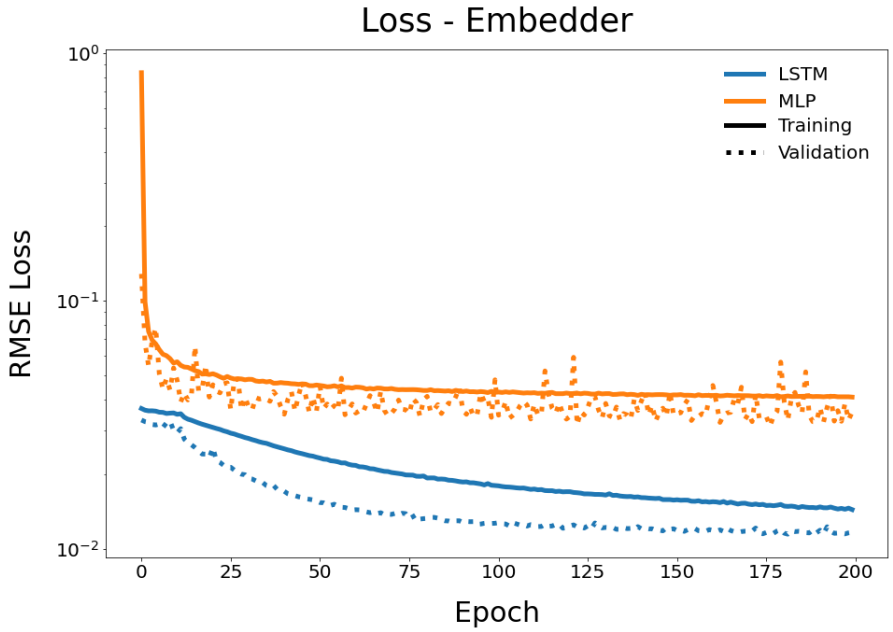

Figure 5.16: The loss of the embedder model shows the typical decline. The LSTM based embedder has a lower training and validation loss compared to an MLP based embedder.

In order to evaluate the embedder, word classification accuracies are calculated on a test set of 225 unseen word tokens and 13 word types, which show a top-1 accuracy for test data from the segment-based approach of 99 % and for test data from the human recordings (Common Voice corpus) of 63 %. Furthermore, we projected the 300-dimensional embeddings for a subset of the data into the 2-dimensional plane with a Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP) embedding[64] and visualized it as a scatter plot (Figure 5.17). The 2-dimensional projection shows that in the subset all log-mel-spectrograms are embedded close to the target.

In summary, the embedder is the second workhorse in PAULE. It connects the acoustic representation to the semantic representation. In contrast to the predictive forward model, the embedder is not only used in the internal loop but in the outer loop as well to connect produced acoustics to the corresponding produced semantics. The advantage of the embedder compared to the predictive forward model is that it can be trained on real-world recordings of human speech and, therefore, is informed by the real human voice.

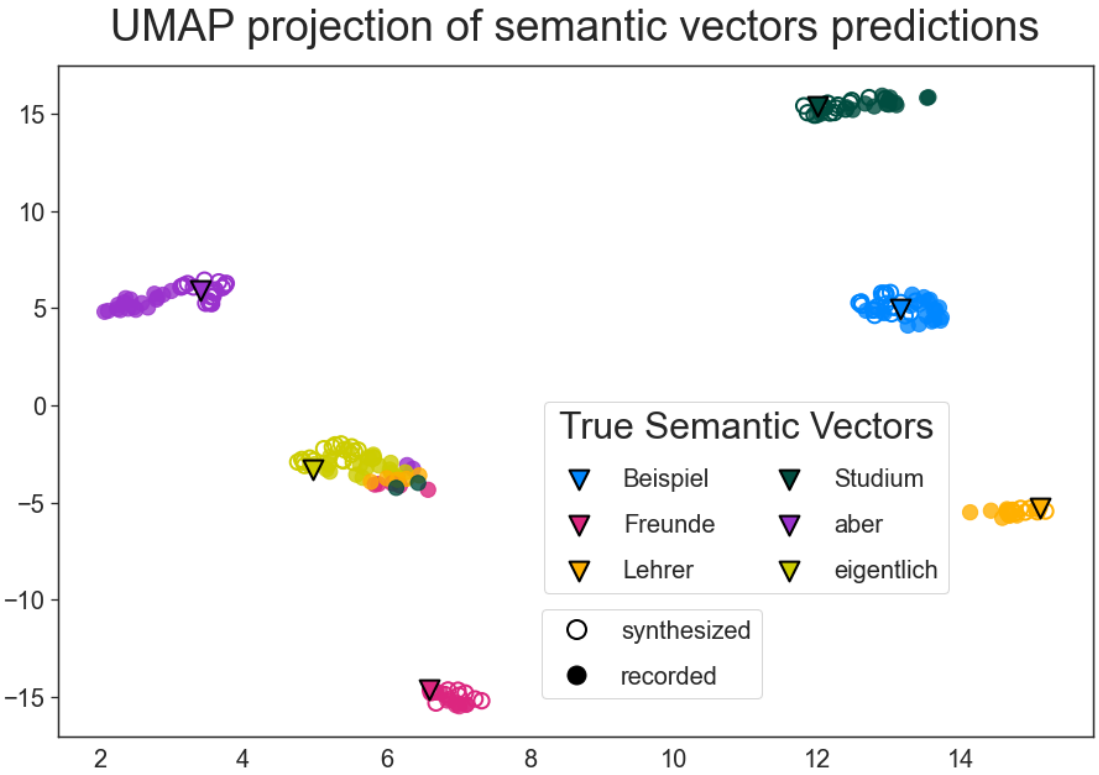

Figure 5.17: This scatter plot shows a 2-dimensional visualization of the 300-dimensional semantic space. Visible are the true semantic vectors as triangles and the embeddings as circles. The produced semantic vectors from resynthesized log-mel-spectrograms are visualized as empty circles whereas full circles show show the positions of the semantic vectors for the human speech recordings. The predictions are done by the embedder model and show that the predictions for the 6 selected word types are all much closer to the target semantic vector than to the other semantic vectors, with the exception of some recordings mapped next to /eigentlich/, which is one of the high-frequency words that can undergo enormous reduction.

5.4.3 Inverse model

The direct inverse model is used to initialize the cp-trajectories that are then subsequently refined (planned) by the internal loop in PAULE in the acoustic-only task (copy-synthesis task) and the semantic-acoustic task. The direct inverse model, therefore, is executed once per planning. It is important that the direct inverse model produces cp-trajectories from a log-mel-spectrogram that is acoustically and semantically close enough to the target acoustics and target semantics so that the gradients of the predictive forward model and the embedder are informative and the successive changes can lower the effort of the cp-trajectories in terms of minimal velocity and minimal jerk while improving acoustic and semantic similarity.

Architecture

The architecture of the direct inverse model is similar to that of the predictive forward model used in 2020[95,93]. The architecture is shown in Figure 5.18 and the code definitions in Listing 5.3. The input log-mel-spectrogram is first modified by five residual convolutional layers, which have filters that span 5 time points and 3 mel-frequency bands. From the enhanced log-mel-spectrogram input velocities (time delta) and accelerations (time delta-delta) are computed and concatenated to the log-mel-spectrogram, which results in a tensor of shape *batch size* $\times$ *180* $\times$ *sequence length*. This tensor is fed into a LSTM-layer with 720 cells. After the LSTM-layers, a linear layer maps the LSTM-activation tensor of shape *batch size* $\times$ *720* $\times$ *sequence length* to the final number of 30 channels. As the output cp-trajectories have a time resolution that is twice that of the log-mel-spectrogram, in the next step, the time resolution is doubled by adding time-steps into the sequence with values that are the mean of the neighbouring time-steps. Now the tensor has the final shape of *batch size* $\times$ *30* $\times$ *twice the sequence length*. In a last step, the network can smooth the values further in time by a stack of five residual convolutional layers that mix five time-steps but do not change the shape. In a final weighting step, the original outputs of the LSTM-layer are concatenated with the smoothed outputs from the convolutional layers. Finally, with a weighting layer the best results are selected from the final output.

The reason for this rather complicated architecture is that noisy log-mel-spectrogram inputs need to be mapped on smooth cp-trajectories. Furthermore, the direct inverse model is used for initialization only and, therefore, never underwent any heavy simplification attempt by us after we got satisfying results.

Listing 5.3: Pytorch definition of the direct inverse model.

```
1 class DirectInverseModel(torch.nn.Module):
2
3     def __init__(self,
4                  input_size=60,
5                  output_size=30,
6                  hidden_size=720,
7                  num_lstm_layers=1,
8                  mel_smooth_layers=3,
```

```
                mel_smooth_filter_size=3,
                mel_resid_activation = torch.nn.Identity(),
                resid_blocks=5,
                time_filter_size=5,
                pre_resid_activation=torch.nn.Identity(),
                post_resid_activation=torch.nn.Identity(),
                output_activation=torch.nn.Identity(),
                lstm_resid=True):
        super().__init__()

        self.lstm_resid = lstm_resid
        self.mel_resid_activation = mel_resid_activation
        self.pre_activation = pre_resid_activation
        self.post_activation = post_resid_activation
        self.output_activation = output_activation

        self.double_sequence = double_sequence
        self.add_vel_and_acc_info = add_vel_and_acc_info
        self.MelBlocks = torch.nn.ModuleList(
            [MelChannelConv1D(input_size, mel_smooth_filter_size)
             for _ in range(mel_smooth_layers)])
        self.lstm = torch.nn.LSTM(3 * input_size, hidden_size,
                                  num_layers=num_lstm_layers,
                                  batch_first=True)
        self.post_linear = torch.nn.Linear(hidden_size, output_size)
        self.ResidualConvBlocks = torch.nn.ModuleList(
            [TimeConvResBlock(output_size, time_filter_size,
                              self.pre_activation, self.post_activation)
             for _ in range(resid_blocks)])

        if self.lstm_resid and len(self.ResidualConvBlocks) > 0:
            self.resid_weighting = torch.nn.Conv1d(
                    2 * output_size, output_size, time_filter_size,
                    padding=2, groups=output_size)

    def forward(self, x, *args):
        if len(self.MelBlocks) > 0:
            x = x.permute(0, 2, 1)
            for layer in self.MelBlocks:
                shortcut = x
                x = layer(x)
                x += shortcut
                x = self.mel_resid_activation(x)
            x = x.permute(0, 2, 1)

        x = self.add_vel_and_acc_info(x)
        output, _ = self.lstm(x)
        output = self.post_linear(output)
        output = self.double_sequence(output)
```

```
58
59          output = output.permute(0, 2, 1)
60          lstm_output = output
61          for layer in self.ResidualConvBlocks:
62              output = layer(output)
63
64          if len(self.ResidualConvBlocks) > 0 and self.lstm_resid:
65              output = [torch.stack((output[:, i, :],
66                                     lstm_output[:, i, :]), axis=1)
67                        for i in range(output.shape[1])]
68              output = torch.cat(output, axis=1)
69              output = self.resid_weighting(output)
70
71          output = self.output_activation(output.permute(0, 2, 1))
72          return output
```

Training & evaluation

The training for the direct inverse model uses the same data set as the predictive forward model, namely the segment-based resynthesis of the Common Voice corpus consisting of 21,175 word tokens and 4,311 word types. The RMSE loss between the output cp-trajectories and the segment-based cp-trajectories is used in the supervised training, complemented with a velocity, acceleration, and jerk loss. Listing 5.4 shows how the loss is calculated. An Adam optimizer with an initial learning rate of 0.001 is used and the training runs over 100 epochs. The training loss dropped from 0.128 to 0.015 (Figure 5.19). The loss is substantially better compared to an MLP model trained on the same data.

Listing 5.4: Pytorch definition of the loss used for the direct inverse model.

```
1  class RMSELoss(torch.nn.Module):
2      def __init__(self, eps=1e-6):
3          super().__init__()
4          self.mse = torch.nn.MSELoss()
5          self.eps = eps
6
7      def forward(self, yhat, y):
8          loss = torch.sqrt(self.mse(yhat, y) + self.eps)
9          return loss
10
11 rmse_loss = RMSELoss(eps=0)
12
13
14 def get_vel_acc_jerk(trajectory, *, lag=1):
15     """returns (velocity, acceleration, jerk) tuple"""
16     velocity = (trajectory[:, lag:, :] - trajectory[:, :-lag, :]) / lag
17     acc = (velocity[:, 1:, :] - velocity[:, :-1, :]) / 1.0
18     jerk = (acc[:, 1:, :] - acc[:, :-1, :]) / 1.0
19     return velocity, acc, jerk
```

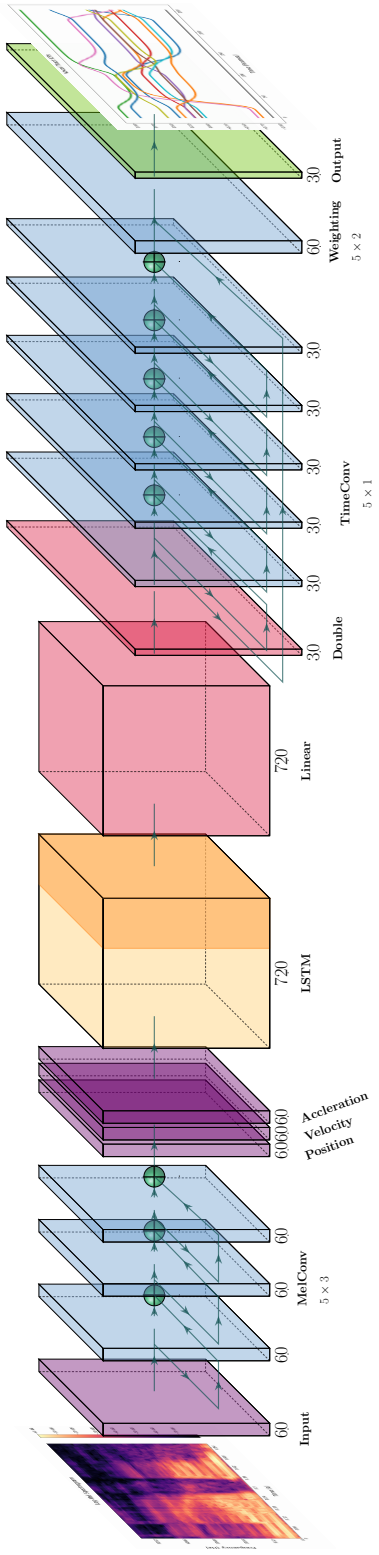

Figure 5.18: The architecture of the direct inverse model is relatively complex with residual convolutions at the beginning and the end of the model, a computation to calculate time deltas (velocities) and time delta-deltas (accelerations) within each trajectory (purple block) and one 720 cell-wide LSTM-layer.

```
20
21
22 def cp_trajectory_loss(Y_hat, targets):
23     """
24     Calculate additive loss using the RMSE of position, velocity, acceleration
25     and jerk.
26
27     :param Y_hat: 3D torch.Tensor
28         model prediction
29     :param targets: 3D torch.Tensor
30         target tensor
31     :return loss, pos_loss, vel_loss, acc_loss, jerk_loss: torch.Tensors
32         summed total loss with all individual losses
33
34     """
35     vel, acc, jerk = get_vel_acc_jerk(targets)
36     vel2, acc2, jerk2 = get_vel_acc_jerk(targets, lag=2)
37     vel4, acc4, jerk4 = get_vel_acc_jerk(targets, lag=4)
38
39     Y_hat_vel, Y_hat_acc, Y_hat_jerk = get_vel_acc_jerk(Y_hat)
40     Y_hat_vel2, Y_hat_acc2, Y_hat_jerk2 = get_vel_acc_jerk(Y_hat, lag=2)
41     Y_hat_vel4, Y_hat_acc4, Y_hat_jerk4 = get_vel_acc_jerk(Y_hat, lag=4)
42
43     pos_loss = rmse_loss(Y_hat, targets)
44     vel_loss = (rmse_loss(Y_hat_vel, velocity)
45                 + rmse_loss(Y_hat_vel2, velocity2)
46                 + rmse_loss(Y_hat_vel4, velocity4))
47     jerk_loss = (rmse_loss(Y_hat_jerk, jerk)
48                  + rmse_loss(Y_hat_jerk2, jerk2)
49                  + rmse_loss(Y_hat_jerk4, jerk4))
50     acc_loss = (rmse_loss(Y_hat_acc, acc)
51                 + rmse_loss(Y_hat_acc2, acc2)
52                 + rmse_loss(Y_hat_acc4, acc4))
53
54     loss = pos_loss + vel_loss + acc_loss + jerk_loss
55     return loss, pos_loss, vel_loss, acc_loss, jerk_loss
```

Figure 5.20 shows some example cp-trajectories that are generated by the segment-based approach (ground truth) compared to the LSTM-based direct inverse model. The direct inverse model learns well to predict smooth trajectories that match the ones produced with the segment-based approach.

The direct inverse model does not solve the one-to-many mapping between one acoustics realized by many different articulatory movement trajectories. It therefore always gives the same cp-trajectories for the same log-mel-spectrogram. In order to solve the one-to-many problem, PAULE applies a planning mechanism with the internal loop. This planning mechanism allows for smooth continuations from the current state and recent past of the VTL cps. Furthermore, the planning allows to find suitable cp-trajectories that result in audio that conveys the right meaning, even if single cp

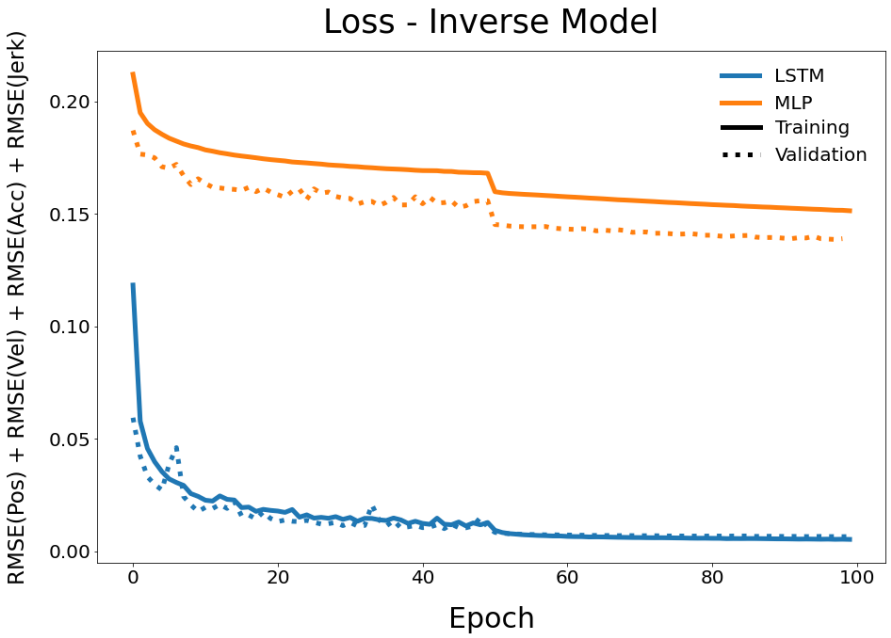

Figure 5.19: The direct inverse model has a substantially lower loss compared to a MLP model. Decreasing the learning rate after 50 epochs helps the model to converge even better.

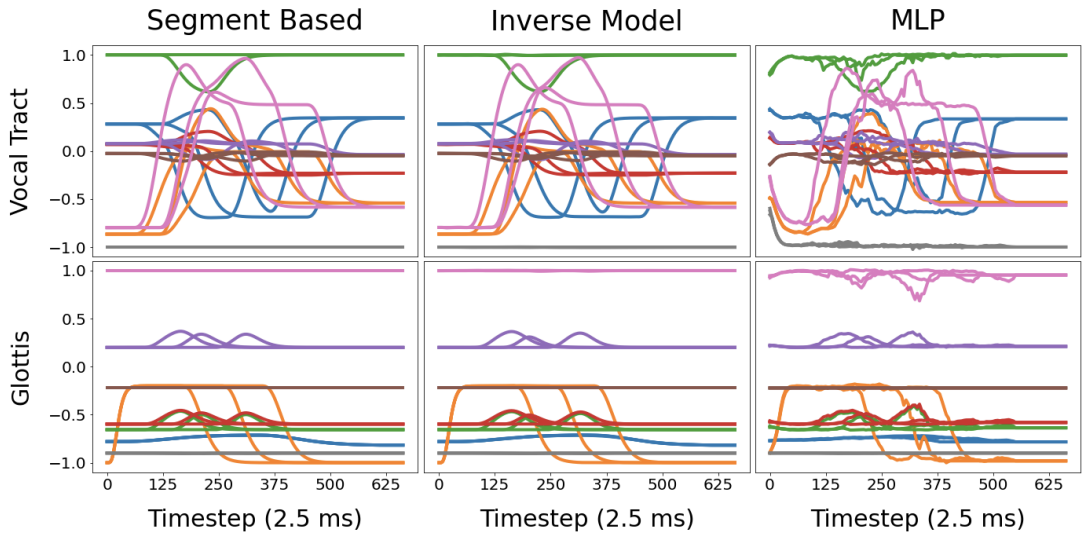

Figure 5.20: The LSTM-based direct inverse model predicts smooth curves that are very close to the ground truth segment-based cp-trajectories. In contrast, the MLP predicts curves that are not smooth, but still get the overall structure correct.

dimensions are restricted. An example for this could be the restriction of the jaw movement to simulate a pen in the mouth situation. This situational and adaptive behavior cannot be done with the direct inverse model alone.

5.4.4 Cp-GAN & mel-GAN

For the semantic-only task the direct inverse model cannot be used for initialization as no target acoustics is given. Therefore two new component models are introduced to initialize the target acoustics and the cp-trajectories directly from the target semantics. These two models are the cp-GAN, which initializes the cp-trajectories, and the mel-GAN, which initializes the target acoustics. The cp-GAN maps a semantic vector and a given duration onto one set of cp-trajectories. Therefore, it is a model that unrolls a fixed length representation onto a variable length time-series data. In PAULE, the cp-GAN can be interpreted as a first initial guess or motor program for a given word meaning. In the following, only the cp-GAN is described, but the mel-GAN has exactly the same architecture except for the last layer where the activations are mapped onto a log-mel-spectrogram instead of the cp-trajectories. A thorough description of the GAN models can be found in Schmidt-Barbo[80].

Architecture

A Generative Adversarial Network (GAN) consists of two separate models that in a game-theoretical sense play a min-max game. One model, the generator, generates (fake) cp-trajectories that should fool the other model, the critic, into believing these are true cp-trajectories. The critic gets the generated (fake) cp-trajectories from the generator and ground truth cp-trajectories and tries to discriminate both classes as well as it can.

For the generator, an adaptive upscaling mechanism is used to unroll the time-series data over the full duration (Figure 5.21 and Listing 5.5). In a first step, a random noise vector with 100 entries is sampled i.i.d. from a standard normal distribution. The random noise vector is concatenated with the target semantic vector. This input vector of shape *batch size* $\times$ *400* is then mapped on a shape of *batch size* $\times$ *1024* with a fully connected linear layer and then reshaped into a shape of *batch size* $\times$ *256* $\times$ *4*. With the reshaping, a time-series dimension is formed with a length of 4 time points. In the next steps, the target duration comes into play by successively upsampling the time-series dimension first to *target duration* / *5* time-steps followed by a residual convolution layer, which uses a filter size of 5 time-steps and 1 channel dimension. The enhanced activations are then batch normalized and passed through a leaky Rectified Linear Unit (ReLU) activation function. This step of upsampling, residual convoluting, and batch normalization is repeated five times with a target upsampling length of *number of steps* $\cdot$ *target duration* / *5*. The fifth and final block results in a tensor with a shape of *batch size* $\times$ *256* $\times$ *target duration*. In the last step, the numbers of channels are mapped from 256 to 30 by a linear layer that is repeatedly applied for each time-step. The final output tensor has a shape of

batch size $\times$ *30* $\times$ *target duration*.[2]

Listing 5.5: Pytorch definition of the generator for the cp-GAN and the mel-GAN.

```
class Generator(torch.nn.Module):
    def __init__(self, channel_noise=100,
                 embed_size=300,
                 fc_size=1024,
                 inital_seq_length=4,
                 hidden_size=256,
                 num_res_blocks=5,
                 output_size=30):
        super().__init__()

        self.fc_size = fc_size
        self.hidden_size = hidden_size
        self.fc_reshaped_size = int(fc_size / inital_seq_length)
        self.fully_connected = torch.nn.Linear(channel_noise + embed_size,
                                               fc_size)

        self.res_blocks = torch.nn.ModuleList(
                [self._block(self.fc_reshaped_size, hidden_size, 5, 1, 2)])
        self.res_blocks = self.res_blocks.extend(
                torch.nn.ModuleList([self._block(hidden_size, hidden_size,
                                                 5, 1, 2)
                for _ in range(num_res_blocks - 1)]))

        self.post_linear = torch.nn.Linear(hidden_size, output_size)
        self.final_smoothing = torch.nn.Conv1d(output_size, output_size,
                                               kernel_size=5, padding=2,
                                               groups=output_size)
        self.output_activation = torch.nn.Tanh()

    def _block(self, in_channels, out_channels, kernel_size, stride, padding):
        return torch.nn.Sequential(
                torch.nn.Conv1d(in_channels, out_channels,
                                kernel_size=kernel_size,
                                stride=stride, padding=padding),
                torch.nn.BatchNorm1d(out_channels),
                torch.nn.LeakyReLU(0.2))

```

[2]In principle, a LSTM-based unrolling architecture could be used as the generator as well. As all other models within PAULE make use of the LSTM-layer, this would be more consistent with the other models. Our experiments with such models, showed promising results, but a LSTM-based generator is substantially more expensive to compute and even the light-weight upsampling generator took by far the most computation power to train in the whole PAULE model. Furthermore, the generator models only need to be executed once during the initialization and optimization and planning of the cp-trajectories within PAULE depends only on the predictive forward model and the embedder. As they are only used once and only in the semantic-only task optimizing this models is by far not as important as for the predictive forward model and the embedder.

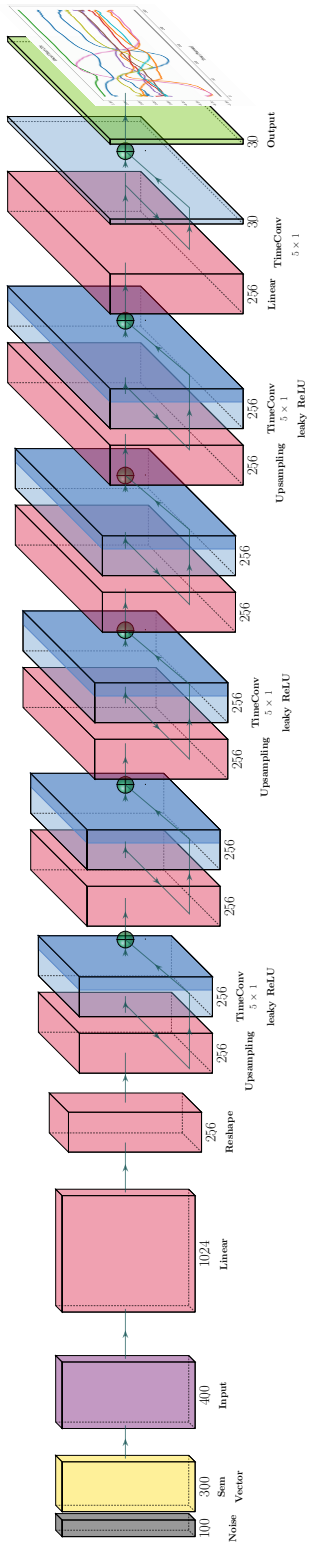

Figure 5.21: The generator model of the cp-GAN. As the generator learns to produce a distribution, it is not only conditioned on the semantic vector as inputs but receives a 100-dimensional noise-vector as well. Furthermore, the generator gets the target duration (number of time-steps) as input. After the semantic vector is concatenated with the noise-vector, the generator upsamples, residual convolutes, and reshapes the inputs in five steps to the target output shape.

```
38      def forward(self, x, length, vector):
39          x = torch.cat([x, vector.unsqueeze(1)], dim=2)
40          output = self.fully_connected(x)
41          output = output.view((len(x), self.fc_reshaped_size,
42                                int(output.shape[-1] / self.fc_reshaped_size)))
43
44          for i, block in enumerate(self.res_blocks):
45              size_ = int(length / (len(self.res_blocks) - i))
46              resizing = torch.nn.Upsample(size=(size_), mode='linear',
47                                           align_corners=False)
48              output = resizing(output)
49              resid = output
50              output = block(output)
51              if i == 0:
52                  if self.fc_reshaped_size == self.hidden_size:
53                      output += resid
54              else:
55                  output += resid
56
57          output = output.permute(0, 2, 1)
58          output = self.post_linear(output)
59          output = output.permute(0, 2, 1)
60          resid = output
61          output = self.final_smoothing(output)
62          output += resid
63          output = output.permute(0, 2, 1)
64          output = self.output_activation(output)
65
66          return output
```

The critic is the counterpart to the generator and plays against the generator model. The critic of the GAN models gets either a batch of generated or ground-truth samples together with the corresponding semantic vectors as input. In the case of the cp-GAN the input contains either generated or segement-based cp-trajectories. For the mel-GAN, the input consists of either generated or recorded log-mel-spectrograms. The task of the critic is to provide a score for whether a given set of cp-trajectories and semantic vectors belongs to the true distribution of ground truth cp-trajectories. This score is designed to approximate the Wasserstein-distance between the distribution of the generated cp-trajectories and the ground truth cp-trajectories. The critic tries to maximize the score between the generated cp-trajectories and the ground truth ones while the generator tries to minimize this score.

In order to calculate the score with the critic (see Figure 5.22 and Listing 5.6), in each time-step of the cp-trajectories the semantic vector is concatenated, which creates a tensor of the shape *batch size* $\times$ *330* $\times$ *number of time-steps*. This tensor has one repeated copy of the semantic vector for each time-step. Only the 30 dimensions that belong to the cps change over time. In the second step, the 330 channels are linearly mapped to 180 channels resulting in a tensor of shape *batch size* $\times$ *180* $\times$ *number of time-steps*. This is

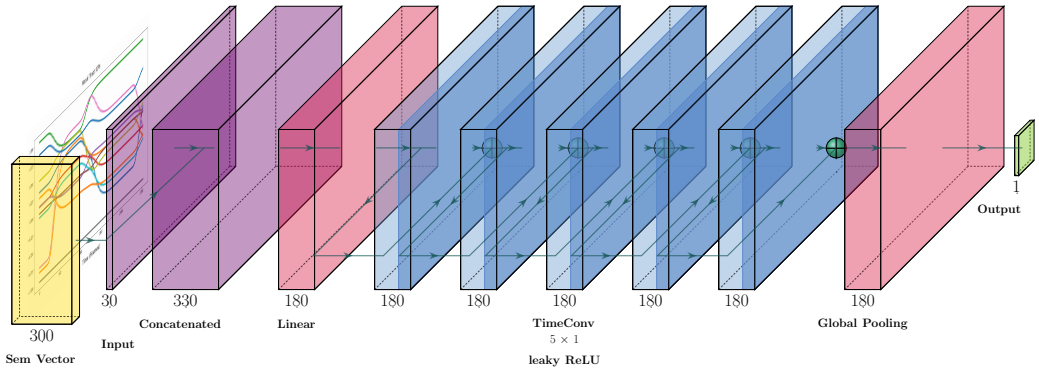

Figure 5.22: The critic of the GAN models gets either a generated batch of cp-trajectories or a segment-based batch of cp-trajectories with the corresponding semantic vectors as inputs (cp-GAN) or a generated log-mel-spectrogram or a recorded log-mel-spectrogram with the corresponding semantic vector as inputs. From these inputs, the critic derives a single score that approximates the Wasserstein-distance between a sample of the generated inputs to the true segment-based or recorded inputs. In order to calculate this single output score, the time-series data is first concatenated with the semantic vector in each time-step and then piped through five residual convolutions. In the final step, the score is derived with a global mean pooling layer.

followed by five residual convolutional blocks with a filter size of 5 time-steps and 1 channel and a ReLU activation function. Over the last activation tensor, global average pooling is used to obtain a single number, the Wasserstein-distance estimate.

Listing 5.6: Pytorch definition of the critic for the cp-GAN and the mel-GAN.

```
class Critic(torch.nn.Module):
    def __init__(self, input_size=30,
                 embed_size=300,
                 hidden_size=180,
                 num_res_blocks=5):
        super().__init__()

        self.inital_linear = torch.nn.Linear(input_size + embed_size,
                                             hidden_size)
        self.res_blocks = torch.nn.ModuleList(
                [self._block(hidden_size, hidden_size, 5, 1, 2)
                 for _ in range(num_res_blocks)])

    def _block(self, in_channels, out_channels, kernel_size, stride, padding):
        return torch.nn.Sequential(
            torch.nn.Conv1d(
                in_channels, out_channels, kernel_size, stride, padding,
            ),
            torch.nn.InstanceNorm1d(out_channels, affine=True),
```

```
20              torch.nn.LeakyReLU(0.2),
21          )
22
23      def forward(self, x, length, vector):
24          x = torch.cat([x, vector.unsqueeze(1).repeat(1, x.shape[1], 1)], dim=2)
25          output = self.inital_linear(x)
26          output = output.permute(0, 2, 1)
27
28          for i, block in enumerate(self.res_blocks):
29              resid = output
30              output = block(output)
31              output += resid
32
33              # average pooling
34          output = output.mean([1, 2])
35          return output
```

Training & evaluation

The cp-GAN is trained on a traing set of the segment-based resynthesis of the GECO corpus containing 46,098 word tokens and 5,444 word types. In one training iteration of the critic, the critic calculates one score for one batch of 64 ground truth cp-trajectories and one score for a batch of 64 generated cp-trajectories. The two scores are subtracted, creating the loss. This loss is optimized using the Adam optimizer, with an initial learning rate equal to 0.0001. Calculating the difference score is done in the way that lower values leads to higher scores for the batch of generated cp-trajectories compared to the ground truth cp-trajectories. Batches are created using same-size batching. Multiple training iterations of the critic are run before a single iteration of the training of the generator is performed. In each training iteration, the generator tries to minimize the score of the critic for the generated cp-trajectories. The generator is never directly exposed to the ground truth data. Figure 5.23 shows the score difference evolution over the 415 epochs. In the beginning, the critic was trained 5 times more than the generator, this was increased to 10 times after 100 epochs, to 20 times after epoch 200, and to 80 times more training after epoch 350. After epoch 365 the number of training iterations of the critic was increased to 100, the maximal possible number on the hardware used. The GAN was trained for 50 more epochs with 100 times more training iterations for the critic compared to the generator.

Evaluating the performance of a generator always usually involves some eyeballing. Figure 5.24 shows a subset of normalized cp-trajectories of the ground truth data compared to three generated trajectories for the word type *Lehrer*. It becomes apparent that the cp-GAN can sample from the underlying distribution and does not just approximate the mean curve. This is even more apparent in the Figure 5.25, where a reference LSTM model only approximates the mean log-mel-spectrogram, while the mel-GAN can generate different variations of the log-mel-spectrogram where each one is more consistent and similar to the exemplar ground truth log-mel-spectrogram and less similar

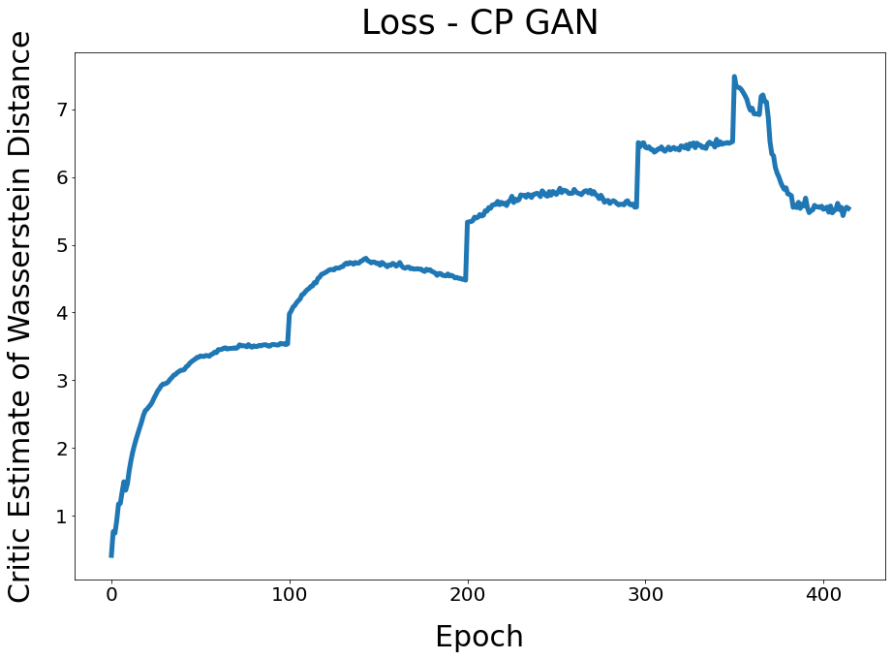

Figure 5.23: The loss of the cp-GAN. The critic estimate is the numeric difference in the critic-score of a generated batch of 64 samples to a batch of 64 true samples, either segment-based cp-trajectories or recorded log-mel-spectrograms. The generator tries to generate samples so that the critic estimate is minimized, whereas the critic learns to maximize the critic estimate. It is important that the critic picks up on meaningful differences between the true samples and the generated samples. In order to ensure this, the number of training iterations in the critic is a multiple of the iterations in the generator. The number of critic iterations is increased at 100 epochs, 200 epochs, 300 epochs, 350 epochs and 365 epochs. These increases in iterations of the critic explain the jumps in the critic estimate.

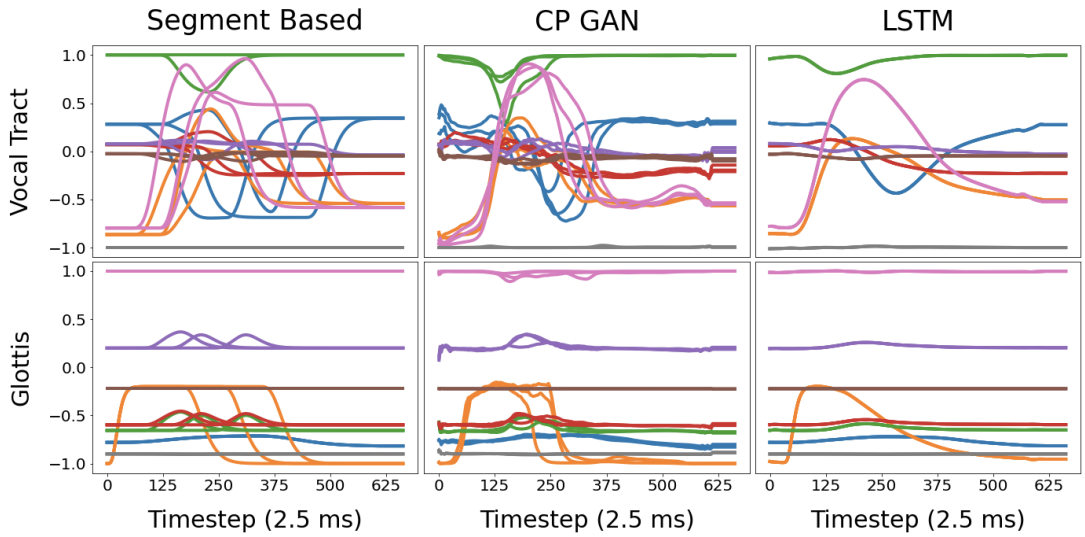

Figure 5.24: Example cp-trajectories generated with a cp-GAN (center panels) and a LSTM-network (right panels) compared to cp-trajectories created with the segment-based synthesis as a reference (left panels). The top panels contain 8 vocal tract parameters for three samples and the bottom panels contain 7 glottis parameters for these three samples. The cp-trajectories shown are randomly sampled from the distribution of cp-trajectories learned by the generator of the cp-GAN. The generated cp-trajectories are not as smooth as the segment-based ones, but show the same overall pattern and are coherent. In the orange glottis trajectories, the changes between the two different states are as sharp as in the segment-based approach. No averaging effect is visible in contrast to an LSTM model trained on the same task.

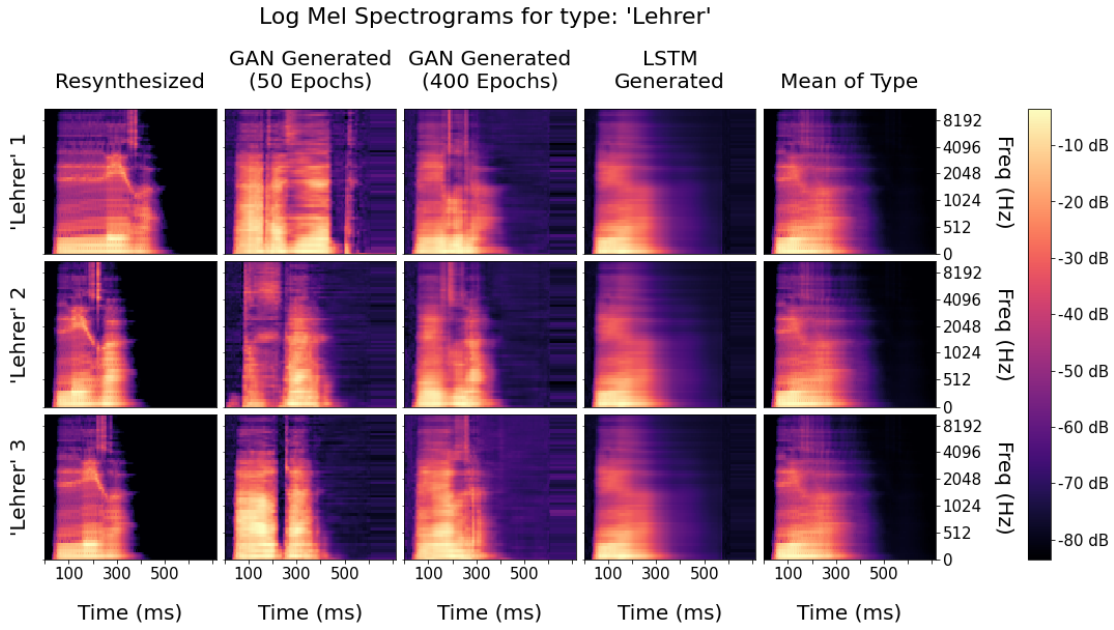

Figure 5.25: Three example log-mel-spectrogram from the resynthesized, ground truth, data compared to generated samples from the mel-GAN after 50 epochs and 400 epochs. For comparison, LSTM-based generations are shown as well that approximate the mean of the type and do not show any variability. The log-mel-spectrogram samples generated by the mel-GAN show a coherent structure and variability in the different samples of the same word type.

to the mean log-mel-spectrogram.

To derive the initial cp-trajectories and the target acoustics (log-mel-spectrogram) GAN models are used and necessary as they achieve to sample coherent exemplars from an underlying distribution without the limitation of predicting mean cp-trajectories. As the planning in PAULE exploits approximated local gradients, it is crucial to start near an optimal solution.

5.5 Baseline models

In order to obtain an idea of what the individual models, but also what PAULE as a framework of articulatory speech synthesis achieves, it is important to constructed with very simple models. The purpose of these baseline models is to show what is possible without extensive calculations. Baseline models also provide a reference value against which more complex models can be compared. This reference value should be outperformed by different configurations of PAULE. In what fallows, a series of baseline models is set up. The next chapter provides detailed comparisons of fully-fledged PAULE models with these baselines.

5.5.1 Schwa-model

To provides a baseline for PAULE as a whole, we defined a *Schwa-model* that outputs constant cp-trajectories for this neutral vowel. If these cp-trajectories are given to the VTL synthesizer a /schwa/ sound is generated. This model can generate every duration by concatenating the neutral cp state for as many time-steps as desired and produces the same response for every target acoustics or target semantics. As it returns the requested cp-trajectories, every evaluation metric can be computed with this baseline model in the same way as the evaluation metric can be computed for cp-trajectories planned by PAULE or any other control model of the VTL.

5.5.2 PAULEmlp

For each of the individual component models within PAULE, we created parallel MLP models (see Listing 5.7). These models assume independence between the time-steps in the time-series. Therefore no auto-correlative dependencies are modeled. These are wide networks containing one ReLU activation function as a non-linearity and serve as a baseline for the LSTM-based models used in PAULE. As we can create an MLP model for each individual model, we can also create a PAULE model consisting of these MLP-based models. This PAULE model is called PAULEmlp as it only makes use of wide linear mappings with a single non-linearity.

Listing 5.7: Pytorch definition of the MLP models for the predictive forward (pred), inverse (inv), and embedding (embd) model.

```
1 class NonLinearModel(torch.nn.Module):
```

```
2       def __init__(self,
3                    input_channel=30,
4                    output_channel=60,
5                    hidden_units=8192,
6                    activation_function=torch.nn.LeakyReLU(),
7                    mode="pred",
8                    on_full_sequence=False,
9                    add_vel_and_acc=True):
10          super().__init__()
11          self.on_full_sequence = on_full_sequence
12          self.add_vel_and_acc = add_vel_and_acc
13          assert mode in ["pred", "inv", "embed"], ("if you want to train a "
14                  "predictive model please set mode to 'pred', for a inverse "
15                  "model set mode to 'inv', and for embedder to 'embd'!")
16          self.mode = mode
17          if self.on_full_sequence:
18              if self.add_vel_and_acc:
19                  self.input_channel = input_channel * 3
20                  self.add_vel_and_acc_info = add_vel_and_acc_info
21              else:
22                  self.input_channel = input_channel
23              if self.mode == "pred":
24                  self.half_sequence = torch.nn.AvgPool1d(2, stride=2)
25              elif self.mode == "inv":
26                  self.double_sequence = double_sequence
27          else:
28              self.input_channel = input_channel * 2
29
30          self.output_channel = output_channel
31          self.hidden_units = hidden_units
32          self.activation_function = activation_function
33          self.non_linear = torch.nn.Linear(self.input_channel, self.hidden_units)
34          self.linear = torch.nn.Linear(self.hidden_units, self.output_channel)
35
36      def forward(self, x, *args):
37          if self.on_full_sequence:
38              if self.add_vel_and_acc:
39                  x = self.add_vel_and_acc_info(x)
40              if self.mode == "embed":
41                  x = torch.sum(x, axis=1)
42          else:
43              x = x.reshape((x.shape[0],1, -1))
44          output = self.non_linear(x)
45          output = self.activation_function(output)
46          output = self.linear(output)
47          if self.on_full_sequence:
48              if self.mode == "pred":
49                  output = output.permute(0, 2, 1)
50                  output = self.half_sequence(output)
```

```
51                 output = output.permute(0, 2, 1)
52             elif self.mode == "inv":
53                 output = self.double_sequence(output)
54         return output
```

5.5.3 LSTM instead of GAN

In order to evaluate the GAN component models, we trained some LSTM based models, which could only create one output per semantic vector input and, therefore, converged to the mean curves in the training data, whereas the GAN models were able to generate coherent exemplars of the underlying distribution.

6 Results

> If you find that you're spending almost all your time on theory, start turning some attention to practical things; it will improve your theories. If you find that you're spending almost all your time on practice, start turning some attention to theoretical things; it will improve your practice.
>
> Donald Knuth

Different versions of PAULE have been evaluated with pilot studies and presented at conferences. These pilot studies are collectively presented in this chapter. The results show that PAULE implements a control model for the VTL that is capable of synthesizing intelligible audio word tokens. Section 6.6 describes a way to systematically and thoroughly evaluate PAULE. Unfortunately, the suggested benchmark is not completed yet and therefore results of the benchmark will be published in the future.

Before showing results that give evidence that PAULE works as intended and is a functioning control model for the VTL speech synthesizer without any symbolic representation in the acoustic or motor domain, it should be noted that no experiments with human listeners have been conducted to evaluate the quality of the synthesized speech. Listening to the synthesized audio ourselves overall gives a mediocre speech quality impression with a synthesized audio that is intelligible most of the time. The speech quality is more naturalistic compared to the mechanically sounding segment-based approach, but less good compared to modern speech synthesis systems.

It is difficult to tell, which limitations in quality are due to the articulatory speech synthesizer VTL and its approximations of the physical processes in the speech production of humans and which are due to an imperfect control model. Additionally, it is important to remember that the results are planned on real human recordings of speech, which are coming from a diverse set of microphones, include some background noise and most importantly come from different speakers. The VTL shape of the oral cavity is fixed to one specific male speaker and, therefore, it is impossible for PAULE to exactly match the target acoustics. All evaluations have been done on new test data that was never used in training or validation. That the evaluation is done on held-out test data is crucial. If the test data would be part of the training data, it would be possible and likely that the individual component models memorize the training sample and have no need to generalize to new samples. Furthermore, the data for the evaluation comes from real human recordings that differ in their statistical distribution to the training data PAULE is trained on. Except for the embedder and the mel-GAN, all component models in PAULE are trained on acoustics that the VTL can produce with the segment-based approach. In contrast to the well articulated speech in the training data, the targets in the

evaluation are spliced out human recordings from natural speech and therefore often highly reduced and barely intelligible on their own. Still, PAULE is able to approximate these targets most of the time in terms of matching the log-mel-spectrogram and the semantic vector closely.

6.1 Loss reduction & classification accuracies

The first and most important metric is how good PAULE achieves meeting its own goals in controlling the VTL. Is PAULE actually able to minimize the error during planning in the synthesized audio of the VTL? As PAULE uses a short-cut or surrogate model with the predictive forward model, loss reduction in the imagined predicted acoustics and predicted semantics is not of interest, but the loss reduction of the error in the produced acoustics and produced semantics compared to the respective targets is. The produced acoustics and produced semantics are the ones where the finally planned cp-trajectories are given to the VTL and then the synthesized audio is converted to a log-mel-spectrogram and a semantic vector. The loss reduction of the produced acoustics and produced semantics is evaluated in the following.

As PAULE can plan cp-trajectories for three different tasks, the loss reduction is shown separately for each task. The three tasks are: the semantic-only task (full generation task), where only a semantic vector and a duration are given; the semantic-acoustic task (supervised generation task), where both a target semantics and a target acoustics are given; and the acoustic-only task (mimicking or copy-synthesis task), where only a target acoustic is given.

Furthermore, in order to explore the effect of the different additive loss components, the planning can either be informed by the semantic loss and the acoustic loss (acoustic-semvec objective), only by the semantic loss (semvec objective), or only by the acoustic loss (acoustic objective). The initialisation (init) refers to the cp-trajectories that are initialized by PAULE either with the direct inverse model or the cp-GAN.

To evaluate the different task objective combinations a small test set of 225 unseen word tokens for 13 word types from the Common Voice is created. The word types belong to high- to mid-frequent words and there are 10 to 20 word tokens per type, which minimizes any word frequency effect in the evaluation. The word-frequency effects is present due to the training of the individual models within PAULE. These individual models are trained on 21,175 word tokens and 4,311 word types, which show a classical word-frequency distribution as depicted in Figure 4.1. Details of the training of the individual models is part of the individual model description in Section 5.4.

Figure 6.1 shows the final semantic RMSE loss between the target semantics and the produced semantics, which is derived by calculating the log-mel-spectrogram for the synthesized audio and then using the embedder to obtain a semantic vector from the log-mel-spectrogram. A smaller loss is a better result. The semantic loss during initialization is as large as the loss for the semantic vectors for the generated log-mel-spectrogram from the mel-GAN and substantially larger compared to the semantic loss derived from the recordings or from the segment-based approach. The loss from the recordings and

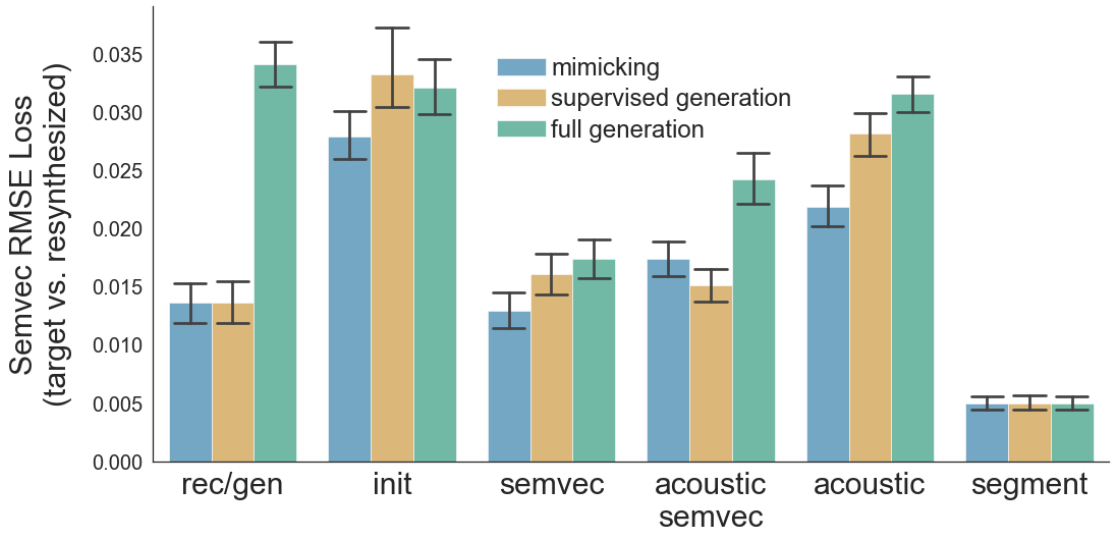

Figure 6.1: The semantic vector loss is shown for three different tasks (mimicking, supervised generation, full generation) along five different optimization objectives. The lower the loss, the better the synthesis. A loss of 0.0 indicates that the synthesized audio is embedded onto the target semantic vector. The error of the recording and generated acoustics (rec/gen; most left bars) indicate the average semantic loss of human recordings and generated samples from the mel-GAN. This is the lowest loss achievable with natural speech given the accuracy of the embedding model. After initialization (init) the average semantic loss is largest (second set of bars from the left). This is the semantic loss of the direct inverse model in the mimicking and of the cp-GAN in the supervised generation and full generation task. The semvec, acoustic-semvec, and acoustic objective are three different ways on how the PAULE model plans cp-trajectories. In the *semvec* objective only the semantic loss and the velocity and jerk constraints are used to drive the planning process. This leads to the lowest semantic loss for cp-trajectories generated by PAULE. In the *acoustic-semvec* objective the acoustics and the semantics are used to inform the planning. The supervised generation task shows that initializing the cp-trajectories from the semantic vector works fine as long as there is a realistic target in form of an audio recording available. Using the generated target acoustics during planning in the full generation task leads to a larger final semantic error. If only the *acoustic* objective is optimized for during the planning, only a slight improvement from the initial semantic loss can be observed. This small improvement is mediated through an improvement in the acoustic similarity (see Figure 6.3). The semantic loss of the segment-based approach is shown to the right. These losses have overall the lowest semantic loss, which is due to the fact that the segment-based approach generates well articulated and highly regular speech and the embedder is trained on these regularities.

the segment-based synthesis are the lower limit what is achievable as a semantic loss. This lower limit origins in limitations of embedder. The planning with PAULE should reduce the semantic loss from the initialization, but the planning cannot reduce the loss below the loss in the segment-based approach. All task objective combinations reduce the loss except for the full generation task using the acoustic objective.

In the semvec and acoustic-semvec objective, where the semantic loss is part of the additive planning loss, the initial loss is reduced to the size of the recording loss. This works except for the semantic-only task (full generation task) in the acoustic-semvec objective, where the generated target acoustics apparently pulls the semantic loss in the wrong direction via the acoustic loss. In the acoustic objective, where no explicit planning is done for the semantics, the loss is reduced by a smaller amount, if at all, as expected. In this last objective no explicit planning on the semantic loss is done in PAULE and the improvements come purely from the improved acoustical similarity.

These results of the semantic loss show that the explicit planning on the semantic loss improve the semantic loss in terms of the finally produced acoustics. Therefore the planning not only reduces the loss in the imagined predicted semantics but also in the produced semantics derived from the synthesized audio. This result is also reflected in a better classification accuracy as shown in Figure 6.2. Now higher values are better values. The classification is done on a test set with mid to high-frequency words and the classifier has to distinguish between 4,311 word types (classes). The small semantic loss in the segment-based approach translates to a very high top-1 accuracy of 99 %. The accuracy of the human recordings is with 63 % substantially lower but still high and comparable to human listeners in the task of single word identification[4]. After initialization, the accuracy is at 12 % to 14 % and planning in the acoustic-semvec objective achieves accuracies of up to 60 %. Note that low-frequency words have not been evaluated yet and it is unknown how well the model performs for these. The results show that high- and mid-frequency words are not confused with low-frequency words most of the time, but does now give evidence how low-frequency words are classified when synthesized with PAULE.

Our interpretation and conclusion from this improvement in top-1 accuracy in a 4,311 classes classification task is that the planning in PAULE with the full additive loss (acoustic-semvec objective) helps to articulate word tokens that can be discriminated better than the ones generated by a direct inverse model or a cp-GAN (*init* objective in Figure 6.2). Assuming that the planning does not find shortcuts in the embedder, that translate to physical simulation of the VTL, it furthermore shows that PAULE can produce intelligible word tokens that evaluate for the mimicking task in the acoustic-semvec objective to be as intelligible as human recordings.

The acoustic loss is the last loss to be investigated in more detail. Figure 6.3 shows the RMSE loss between the target acoustics (recording or generated log-mel-spectrogram) and the produced acoustics. The loss to the recording is zero as this is the reference as well. Overall, it can be seen that applying the acoustic loss during planning in the acoustic-semvec and the acoustic objective substantially reduces the acoustic loss and the acoustic similarity is in the end higher between the produced acoustics from PAULE

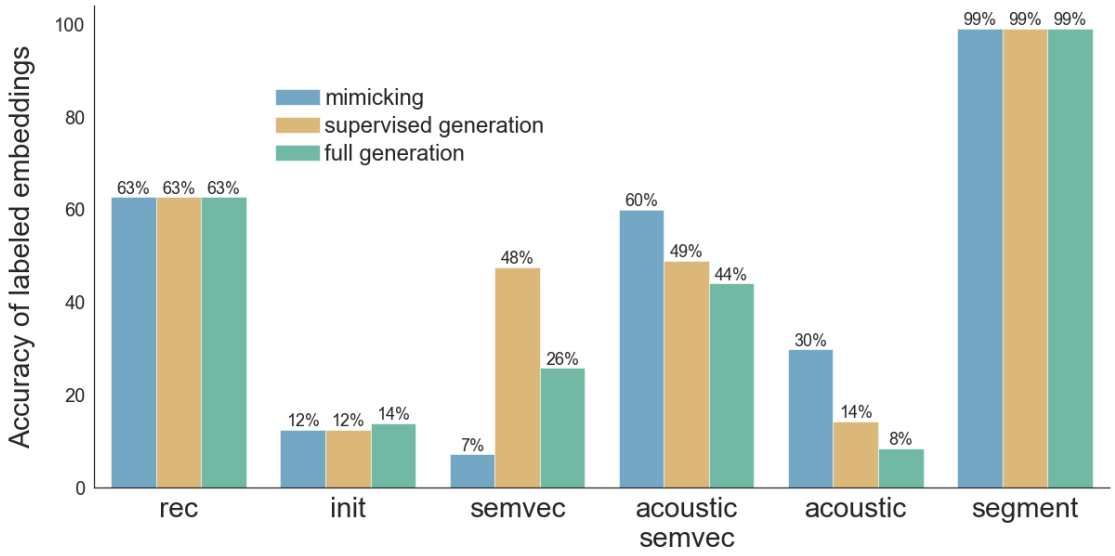

Figure 6.2: Top-1 classification accuracy (higher is better) of a test data set containing 13 word types and 225 word tokens. The 225 word tokens are classified by a classifier that can distinguish between 4,311 word types. The word tokens for classification are synthesized using PAULE with three different objectives in three different tasks. Planning on the acoustics and the semantics jointly (*acoustic-semvec* objective) yields the best result in all three tasks. A detailed explanation of all the conditions can be found in the caption of Figure 6.1.

compared to the initial cp-trajectories and the segment-based approach.

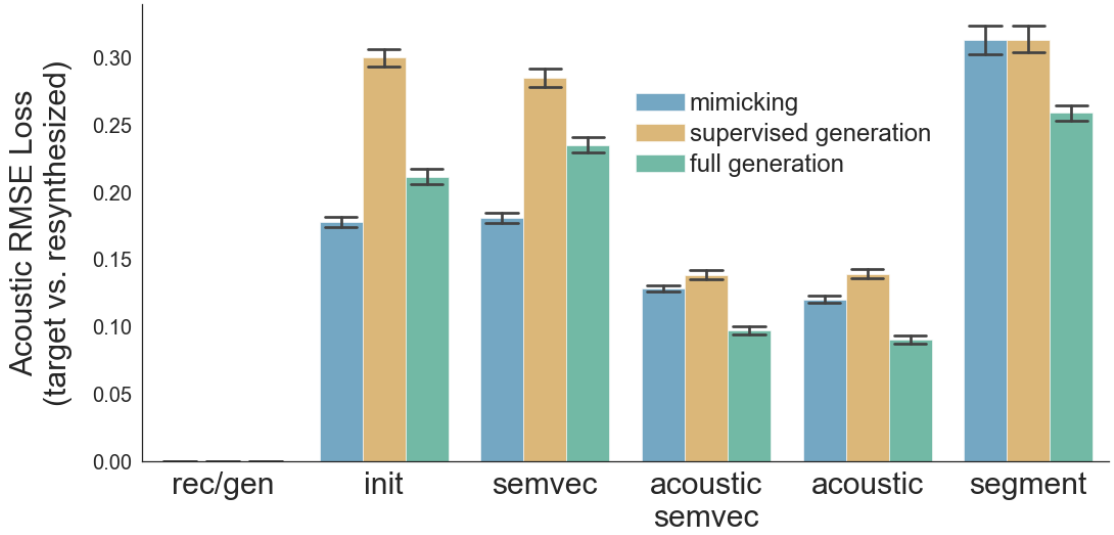

Figure 6.3: The RMSE for the acoustic loss shows that the recording and generated acoustic has the best results as it is compared to itself. Note that in the full generation task (semantic-only task) the target acoustics is a log-mel-spectrogram generated by the mel-GAN. The segment-based approach shows the worst acoustic loss together with the initially produced acoustics (init) in the supervised generation task. The planning improves the acoustic loss equally in the acoustic and acoustic-semvec objective but fails to improve the acoustic similarity if only planned on the semantic loss (semvec objective). A detailed explanation of all the conditions can be found in the caption of Figure 6.1.

In summary, cp-trajectories planned by PAULE lead to synthesized word audio tokens that reduce the semantic and acoustic loss compared to the synthesized word audio tokens from the initial cp-trajectories. Furthermore, the semantic embeddings of the synthesized word audio tokens are more often classified as the correct word by the embedder. All parts of the additive planning loss are needed and the planning finds a compromise between similar acoustics and similar semantics in the final produced audio. As the planning is a computationally intensive, iterative, step-wise process, planning for more iterations is expected to even improve results further. Classification results presented here are based on an automatic classification by the embedder. In order to rule out that the planning in PAULE finds shortcuts in the embedder, i. e. that PAULE improves classification accuracy without improving the synthesis quality for human listeners, a listening and evaluation experiment will be required in the future. Single word classification for spliced-out natural speech is a difficult task for humans, therefore this experiment should be conducted after PAULE is able to synthesize whole sentences. Keep in mind that aim of building PAULE is to synthesize naturalistic, conversational speech and not read-out highly articulated speech like the segment-based approach. Therefore somehow unintelligible single words are a desired output, as long as they

become intelligible in context. Figure 6.4 shows how the iterative planning improves the acoustics step-by-step in the semantic domain.

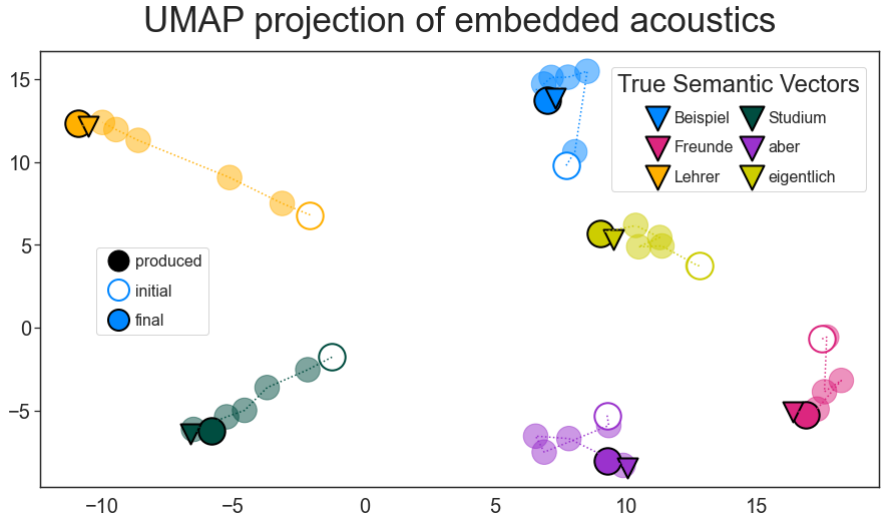

Figure 6.4: The planning in PAULE moves the semantic embedding of the synthesis closer to the intended target semantic vector. This is visualized by the trajectories in a 2-dimensional UMAP projection of the 300-dimensional semantic space.

6.2 Conditioning on past cp-trajectories

A first step to plan full sentences with PAULE involves the necessity to condition on already executed cp-trajectories that cannot be changed anymore. The following cp-trajectories should smoothly connect to the past cp-trajectories. As PAULE has been designed as a predictive, adaptive model from the beginning, this conditioning on past cp-trajectories is straightforward to implement. The main idea is to concatenate the past cp-trajectories before the initialized cp-trajectories. The acoustic error is kept zero for the time-steps in the log-mel-spectrogram that correspond to the past cp-trajectories but the velocity and jerk loss, as well as the semantic loss, are computed over the extended cp-trajectories. During backpropagation, the error signal for the past cp-trajectories are set to zero so that no corrections are applied at these time-steps.

Figure 6.5 shows that the past cp-trajectories concatenated to the initial cp-trajectories show a sharp jump between the time points where the concatenation takes place (between time-step 8 and 9 in the beginning). The optimized or planned cp-trajectories show no jump anymore. Therefore, the planning achieves to smooth out the cp-trajectories while improving the semantic and acoustic loss. The difference plot shows that no corrections are applied in the past cp-trajectories (first eight time-steps), but substantial corrections are done in the rest of the trajectories.

Note that this smooth gluing of cp-trajectories works with PAULE at any point in

time. This gluing is only a first necessary step for synthesizing full utterances. Not solved is what is the proper iterative implementation of the semantic embedding vector representation and how to integrate them into PAULE. A design choice has to be made here as semantic vector representations exist for contextualized words as well as for whole sentences or even paragraphs or sections. As the right choice for a semantic vector representation is only one important question of a few for a proper implementation of full sentence synthesis with PAULE, the full sentence synthesis is left for future work.

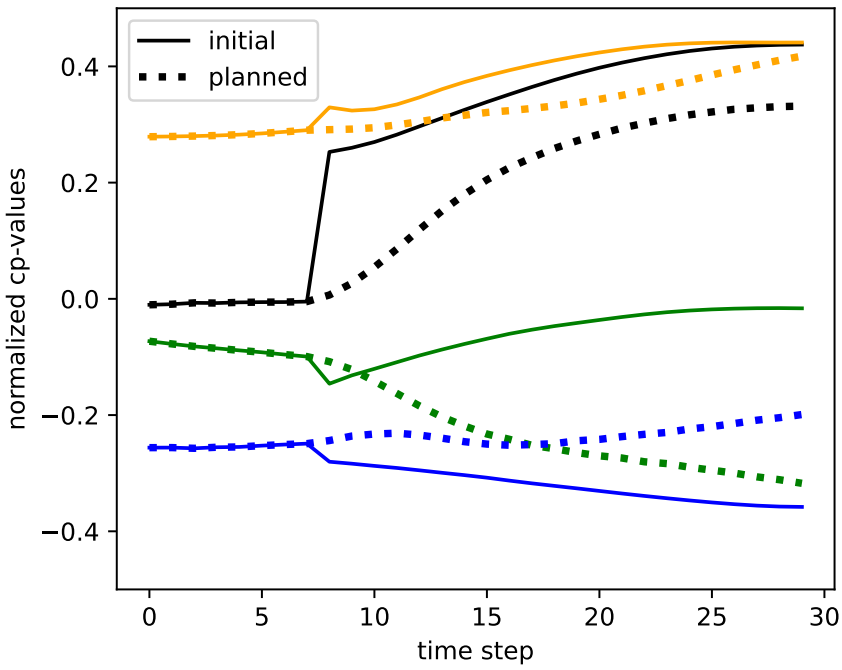

Figure 6.5: With PAULE it is possible to condition the planning on a recent past of cp-trajectories and find solutions that smoothly continue from this past. The direct inverse model introduces a discontinuity (jump) in the initial cp-values, which is visible between time-step 8 and 9. Through the planning by PAULE the discontinuity is removed. The first 30 time-steps of four cp-trajectories are shown.

6.3 Anticipatory coarticulation

To show that PAULE can model anticipatory coarticulation over consonant boundaries, a small data set of artificial /baba/, /babi/, /babu/ utterances at different speaking rates was recorded. In the recording session not only the audio was recorded but additionally, the mid-sagittal plane of the tongue was measured using ultrasound. Recordings were conducted on a single speaker. Results from the ultrasound recording are presented and discussed in the next section.

As the closure to create the /b/ sound is effected by the lips, the tongue and jaw have a high degree of freedom to anticipate the second vowel. This anticipatory behavior can already be seen at the end of the first /a/ sound and is systematically different depending on the following vowel. If the following vowel is an /a/, i. e. the /baba/ condition, the first formant (F1; the first resonance frequency) is shifted to a higher pitch compared to the /babi/ and /babu/ condition, whereas the second formant (F2) is at the same pitch as in the /babu/ condition but lower compared to the /babi/ condition. Figure 6.6 shows the formant shifts for the last 20 milliseconds before the end of the first /a/ for F1 and F2.

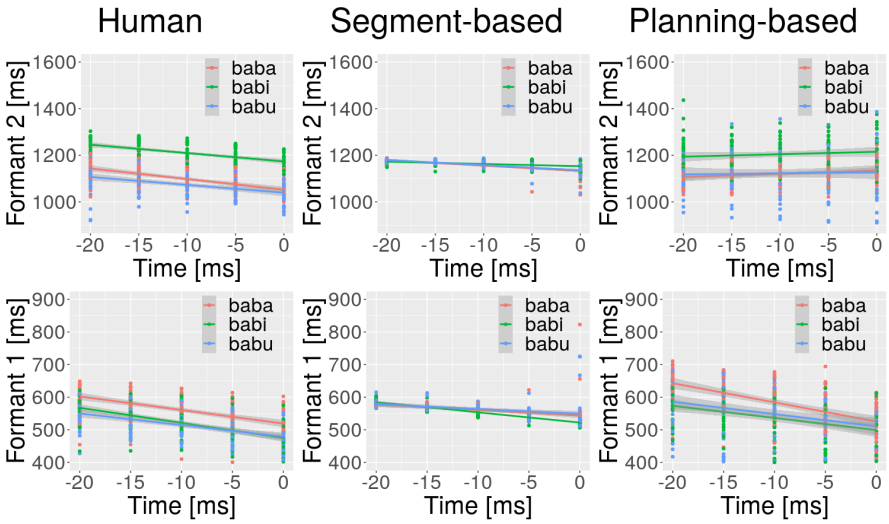

Figure 6.6: Formant shifts of the first /a/ in the PAULE resynthesis (planning-based) resemble human formant shifts in artificial /baba/, /babi/, /babu/ articulations. The fully automated segment-based synthesis fails to capture the formant shifts, which are dependent on the upcoming vowel.[96]

The formant shifts recorded from the human speaker can be explained by anticipatorily raising the jaw and tongue to prepare for the following vowel. In the segment-based approach, these formant shifts are not present, but the planning-based resynthesis with the PAULE model shows the expected formant shifts[96]. The PAULE model is used with an acoustic-only task using only the acoustic objective on single full word utterances. The formant change in time is going in the wrong direction in the PAULE resynthesis and the variability is substantially larger compared to the human recording. This variability might be reduced, if the simulations are repeated with the up-to-date version of PAULE, which is left for future work. Further note, that manual changes to the segment-based approach might also lead to the expected formant shifts visible in the human recording.

6.4 Mid-sagittal ultrasound data

As PAULE is a control model for an articulatory speech synthesis system, it is possible not only to look at features in the resynthesized audio but additionally, to look on the articulatory movements and compare them to recordings by humans. As the mid-sagittal plane with ultrasound for the /babababibabu/ data was recorded, a method to compare the ultrasound recordings to the VTL tongue movement was developed.

In the ultrasound image, the tongue surface is visible as a bright reflective band and it is possible to extract the tongue contour with signal and image processing methods. This reflection relies on the material difference between the tongue and the air above it and the ultrasound head needs a straight line through the tissue to the tongue without any air in between. During speaking this is not always the case as the tongue tip might be so frontal that an air pocket is between the connecting line of the ultrasound head, which is placed below the jaw, and the tongue tip. Furthermore, the reflection vanishes as well, if the tongue is pressed against the palate so that no air is above the tongue.

There are ideas on how to mitigate this problem, but for this analysis, the highest point of the tongue was extracted as a workaround. This measure does not need a registered tongue contour and does not rely on the full visibility of the tongue. In order to extract the highest point, a tongue contour was fitted to the time points of interest and the highest value was calculated. This is then compared to the highest value of a mid-sagittal tongue contour exported from the VTL synthesis (see Figure 6.7).

As the ultrasound head is strapped to the jaw and tilted slightly, to point to the back of the head, the contours exported by the VTL are rotated onto the lower teeth and tilted in the same way as the ultrasound head. In our analysis in 2021, the tongue raising and lowering was not as pronounced as in humans[96]. Note, that the formant shifts are replicated through the same cp-trajectories (see section above). One reason might be that the formant shifts are achieved by other means and therefore PAULE might control the VTL qualitatively different to how a human controls their speaking organ. Another reason might be due to the high amounts of noise in the cp-trajectories produced by the version of PAULE used in 2021.

All the source code to export and extract the highest tongue point in the mid-sagittal plane of the virtual VTL tongue exists and will be integrated into ARTICUlatory speech synthesis BENCHmark (`articubench`) (see below). Further investigations and a statistically more powerful analysis is needed in the future to clarify, if the coarticulations is achieved by tongue raising and lowering or by different means and therefore if PAULE controls the vocal tract in this aspect qualitatively different to humans.

6.5 EMA data

In contrast to the unregistered tongue contour lines of an ultrasound measurement, electromagnetic articulography (EMA) measures registered tongue, lip, and jaw movements. This can be done with a high time resolution of up to 1,000 Hz. In order to show a proof of concept and elaborated on different ways of comparing EMA recordings in humans

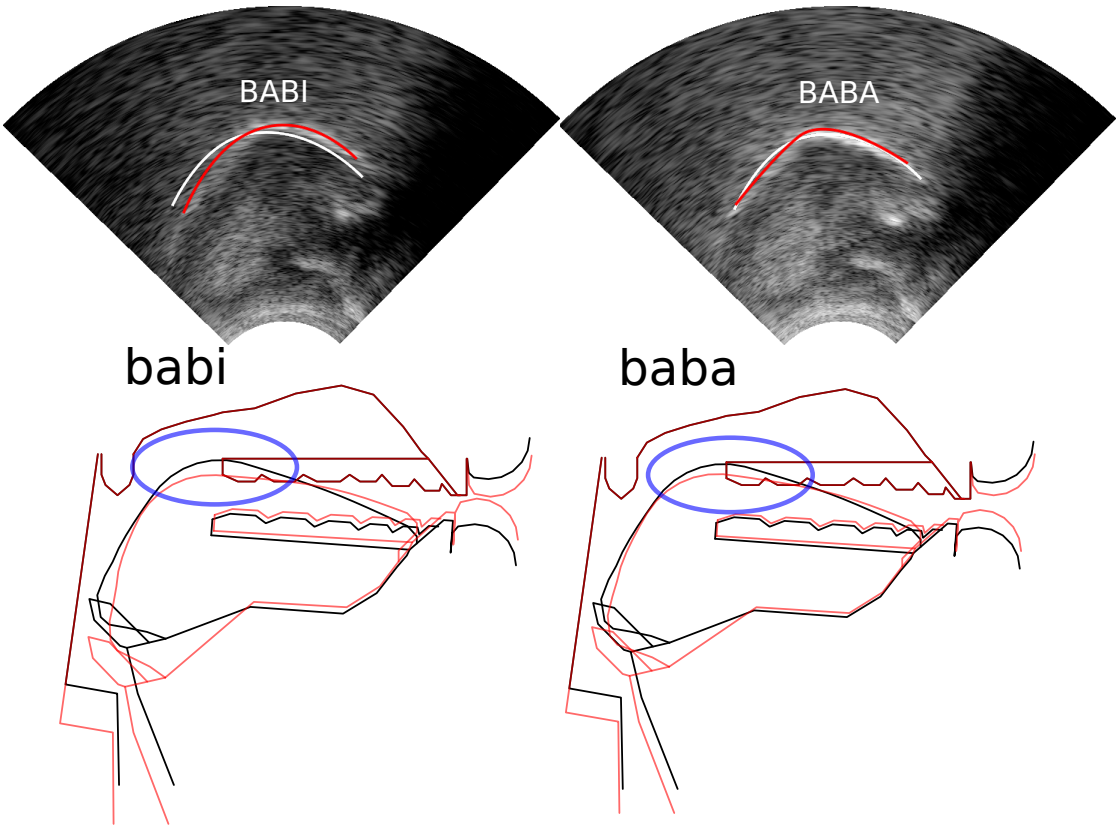

Figure 6.7: Top panel: Tongue position of the human tongue in the ultrasound recording in the middle (white) and at the offset (red) of the first /a/ in /babi/ and /baba/. An anticipatory raising of the tongue body is visible in /babi/ compared to /baba/.
Bottom panel: Tongue position of the simulated VTL tongue in the middle (black) and at offset (red) of the first /a/ in the resynthesized /babi/ and /baba/. A small anticipatory raising of the tongue body is visible (blue ellipses). The statistical analysis indicates that the articulatory tongue raising in the resynthesis models might not be present.[96]

with virtual simulated EMA recordings on the VTL articulators, an API function call was exposed as part of this thesis to select arbitrary nodes in the mesh of the 3-dimensional VTL components end export them with a target sampling rate.

A key challenge is to transform the data into a common reference space, which shares an origin and orientation and potentially normalizes the measured distances to the sizes of the oral cavity of individual speakers. Furthermore, as each EMA sensor is measuring a 3d-coordinate, it has to be decided if the analysis should be restricted to a specific dimension like the front-back movement or the up-down movement or if all three dimensions should be taken into account. One way of aligning the data is to shift the origin of the coordinate system to the maximal value in each coordinate for each individual speaker. Distances can be either kept fixed on a common physical distance like Millimeters or scaled to the relative change to the minimal value in each axis for each individual speaker.

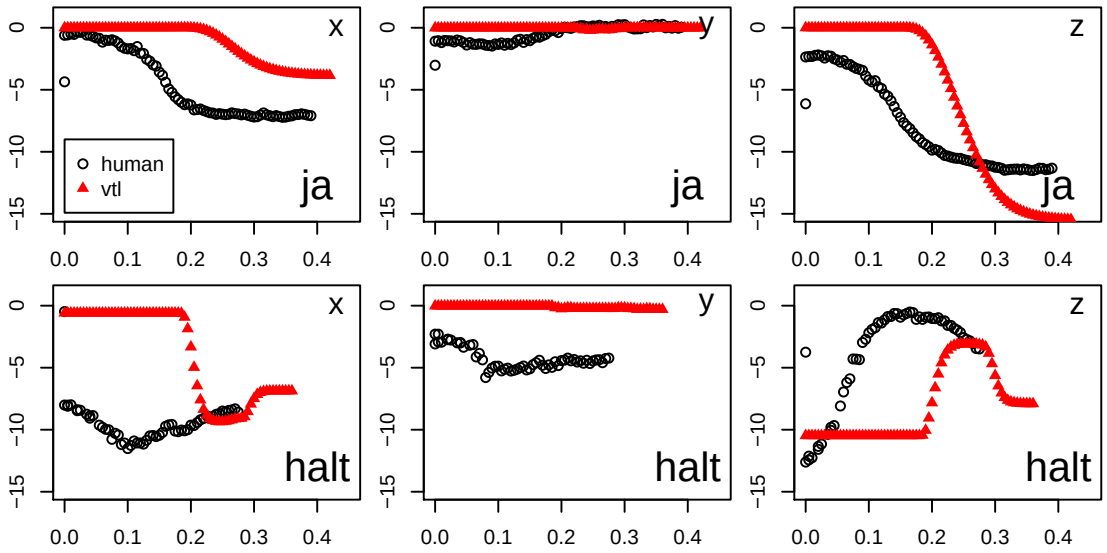

Figure 6.8: Trajectories of the tongue body sensor in *back-front*-direction (x, left panels), in *left-right*-direction (y, middle panels) and in *low-high*-direction (z, right panels) for the word *ja* (top panels) and the word *halt* (bottom panels). The black circles show measured EMA data from one subject and are compared in each plot to the VTL simulated trajectories of a matched sensor (in red triangles). Time is given in seconds, distances in Millimeters. [95]

Some fundamental explorations on differently long series of EMA data have been conducted [95]. For this, a subset of the KEC was used. Figure 6.8 shows the x, y, z coordinates of one /ja/ and one /halt/ articulations trajectory of a single matched sensor. The trajectories show similar time courses, but for some differ in the timings of the transitions. Defining a similarity measure on the full trajectories especially when they vary in duration is challenging. Relying on time points of interest, as it is done for the ultrasound data, can be more informative for some research questions and circumvents

the necessity to deal with different durations. It is planned to add a comparison between recorded EMA data from the KEC and simulated EMA data by the VTL into `articubench`.

6.6 Articubench

The goal of ARTICUlatory speech synthesis BENCHmark (`articubench`)[1] is to bundle semantic, acoustic, and articulatory evaluations of PAULE, so that simulation experiments with PAULE are reproducible. Furthermore, `articubench` is designed in such a way that it is straightforward to compare different control models of the VTL along different evaluation metrics. As evaluation metrics, several scores are calculated. Each score ranges from 0 to 100 and contributes equally to a total score[91].

We set up the Schwa-model as a baseline reference control model. The Schwa-model continuously outputs the cp-trajectories for the /schwa/ sound, which is generated within the VTL by the neutral settings of the cp-trajectories. Scores are normalized to the Schwa-model where feasible so that the Schwa-model has a score of 0. A resynthesis that is perfectly on target is defined to have a score of 100: Such a resynthesis matches exactly the target acoustics or the target semantics.

Scores are separated into three groups: the articulatory, the acoustic, and the semantic group. In the articulatory group the tongue height position is compared to ultrasound recordings or baseline simulations with the *tongue height* subscore. The *EMA score* compares tongue tip and tongue body sensors of a real or simulated EMA recording against EMA recordings simulated with the VTL. The last articulatory score is the *velocity and jerk subscore*. This score evaluates how low the peak velocity and peak jerk is compared to an average calculated from the segment-based approach. In the acoustic group the *loudness subscore* evaluates the loudness envelope of the resynthesized signal to the target acoustics and the *spectrogram subscore* calculates the score from the point-wise differences in the log-mel-spectrogram. In the semantic group the *distance subscore* measures the distance between the target semantics and the produced semantics and the *rank subscore* calculates the classification rank along 4,311 classes with the embedder. The theoretically best total score to achieve is 700 as there are seven subscores.

`articubench` comes with three different tasks and is planned to have three different sizes. The three different tasks are the *acoustic-only task* (copy-synthesis), where only a target acoustics is given, the *semantic-only task*, where a semantic vector and a duration is given, and the *semantic-acoustic task*, where a semantic vector and a target acoustics are given to the control model. The planned three different sizes are the following: A *tiny* version to validate that a control model can be executed with the benchmark, a *small* version that provides an overall score that has more statistical power and a *normal* version that can give additional insight into the substructure of the scores and how the control model performs along different language statistics such as the word frequency. An overview of the tasks and the different metrics is shown in Figure 6.9.

In addition to the Schwa-model the segment-based synthesis model of the VTL was

[1] `https://github.com/quantling/articubench`

enabled to solve the three tasks. For the acoustic-only task the acoustics is first classified with the embedder. The grapheme string of the classified word type is transformed into a phoneme string with the grapheme to phoneme model of the MFA. In a next step the phoneme string is aligned with the MFA to the target audio. The resulting phonemes and corresponding durations are then used as inputs for the segment-based approach. In the semantic semantic-acoustic task the classification step is skipped and instead the given target semantic vector is used to find the closest word type in the classification of the embedder. This is followed up with the grapheme to phoneme transformation and alignment. In the semantic-only tasks phone durations are derived from the average phone duration in the Common Voice corpus, which are prolonged or shortened to match the overall target duration of the word.

6.6.1 Results for tiny data set

Integrating all the analysis scripts of the last years into a fully automated score calculation turned out to be more time-consuming than originally expected. Therefore, only results for the tiny data set in `articubench` are presented here. These results compare four different control models along the three different tasks.

Table 6.1: Preliminary total scores of the tiny version of `articubench` for four different control models.

model	task	score_total
PAULE	acoustic-only	329
	semantic-acoustic	331
	semantic-only	198
PAULEfast	acoustic-only	281
	semantic-acoustic	182
	semantic-only	131
segment	acoustic-only	150
	semantic-acoustic	136
	semantic-only	231
schwa (baseline)	acoustic-only	114
	semantic-acoustic	114
	semantic-only	114

Table 6.1 shows the total score of the results for the tiny data set. The PAULE model uses 10 outer loop iterations and 25 internal loop iterations and requires a wall time of

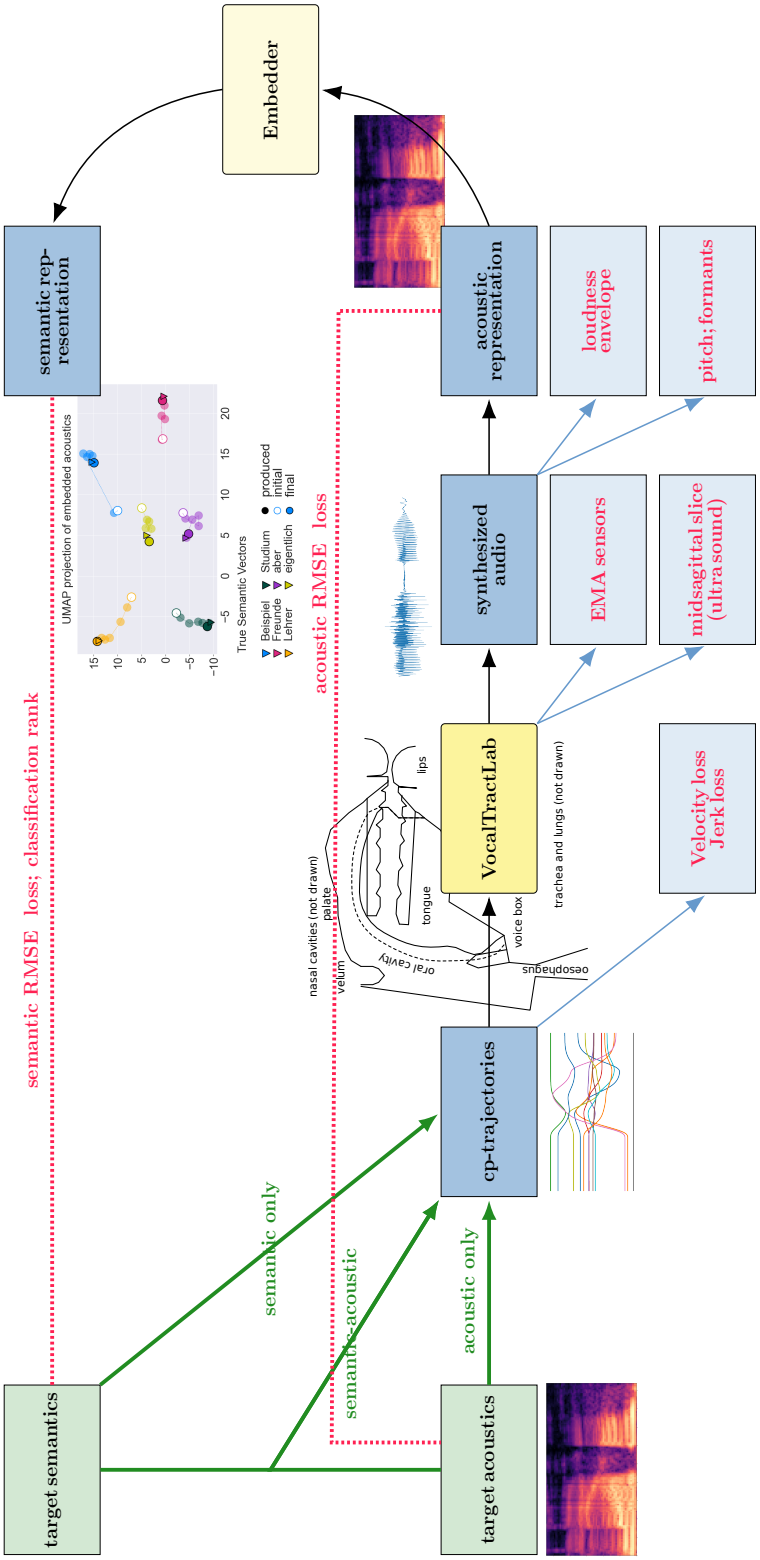

Figure 6.9: The benchmark `articubench` distinguishes between the three different tasks: *semantic-only task*, *semantic-acoustic task*, and *acoustic-only task*. The tasks are denoted in green. The control model tested in `articubench` will result in *cp-trajectories* that solve the task at hand. From these final cp-trajectories, three groups of scores are calculated: *articulatory scores* from the *cp-trajectories* themselves and the associated tongue movements, *acoustic scores* from the produced acoustics, and *semantic scores* from the produced semantics.

30 minutes. PAULEfast only uses 5 internal loop iterations and no outer loop iterations and requires a wall time of 7.20 minutes. The segment-based synthesis uses the MFA for alignment and completes the benchmark in 9 minutes wall time. The baseline Schwa-model, unsurprisingly, requires only 2.34 minutes wall time.

As expected, the baseline Schwa-model shows no differentiation between the tasks and has a constant total score of 114. The Schwa-model as a baseline has the worst score of all models. The segment-based synthesis is the second worst and shows the best results in the semantic-only task with a score of 231. This score is higher than the worst total score of the PAULE control model in the semantic-only task (198). The highest scores are achieved by the PAULE control model. The semantic-acoustic task has the highest score, which is broken into its subscores in Table 6.2.

The total score, which is achieved by the PAULE model in the `articubench` benchmark can be split into its subscores. These subscores are grouped into three groups of articulatory scores, acoustic scores, and semantic scores. As the total scores is the sum of the subscores and all subscores are normalized to be between 0 and 100 an inspection of the subscores gives an impression on where the PAULE model performs well, and where is still room for improvement. The subscores in Table 6.2 can be interpreted the following: In the semantic-acoustic task the PAULE model performs on average in the articulatory score group. The tongue height is more similar to the target tongue height compared to the Schwa-model (score of 51) and the peak velocity and peak jerk is comparable to the segment-based average (score of 0). In the acoustic score group results are good. The loudness envelope is matched (score of 44) and the point-wise differences in the log-mel-spectrogram are substantially better compared to the Schwa-model (score of 47). The best performance is shown in the semantic score group with a subscore for the semantic distance to the target semantics of 87 and a resynthesis that is correctly classified, which results in the best possible rank score of 100. The comparison to the EMA data is not yet fully integrated into `articubench` and therefore not computed and presented here.

6.7 Environmental costs

The final section of Chapter 6 estimates the environmental costs associated with creating and using the PAULE model. The presented version of PAULE needs around 400 kWh of electricity to be fully trained from scratch on the data sets presented in this thesis. The main energy consumption can be traced back to training the GAN models. This accounts for 95 % of the energy consumption or roughly 380 kWh. Training the predictive forward model and the direct inverse model requires around 2 kWh each and the embedder uses around 8 kWh of electricity. The predictive forward model was trained by far most often as a range of different architectures were implemented and tested here. Assuming it was trained 50 times, the energy consumption equates to an additional 100 kWh. It is very likely that not more than 1,000 kWh of electricity were used to develop PAULE over the last seven years. This is less than a single person on average uses in a single year for living in Germany (8,800 kWh). This 8,800 kWh of energy are the energy for heating

Table 6.2: Subscores for the semantic-acoustic task in the tiny version of `articubench` for control model PAULE.

subscore	score value
score_total	331
score_articulatory/tongue_height	51
score_articulatory/ema	NaN
score_articulatory/velocity_jerk	0
score_acoustic/loudness	44
score_acoustic/spectrogram	47
score_semantic/distance	87
score_semantic/rank	100

(6,200 kWh) and electricity (2,600 kWh) in private households averaged onto a single person estimated in for the year 2019[2].

Planning a single word with an average duration of 0.7 seconds requires around 20 minutes on a laptop (130 Watt) resulting in the consumption of around 0.04 kWh per planned word. Therefore, the evaluation data set used for the word classification evaluation consisting of 225 word tokens and used in three tasks and along three objectives required around 81 kWh of electric energy. This is a substantial but not extremely high value, even if it makes the planning of large corpora expensive and time-consuming and is more costly compared to a direct inverse model or the segment-based approach.

Overall, the environmental costs of the PAULE model are substantially lower compared to traveling and living costs. These small environmental costs could be achieved by having relatively small training data sets and intelligently optimizing single models and restraining from grid searches through large model architecture spaces.

[2] `https://www.destatis.de/DE/Presse/Pressemitteilungen/Zahl-der-Woche/2022/PD22_09_p002.html`

7 Discussion

> Le but de la dispute ou de la discussion ne doit pas être la victoire, mais l'amélioration.
>
> Joseph Joubert

The Predictive Articulatory speech synthesis Utilizing Lexical Embeddings (PAULE) framework is a predictive control model for the articulatory speech synthesizer VTL. PAULE allows to copy-synthesize a target acoustics by planning cp-trajectories that are the inputs to the VTL and that result in a 44,100 Hz audio file when given to the VTL. The target acoustics is not limited to speech sounds that can be produced by the VTL but can be any human speech recording, even if resynthesis results vary and are dependent on the quality of the target acoustics. Furthermore, PAULE allows synthesizing speech from target semantics, approximated by a 300-dimensional fastText vector, in combination with a duration of the desired resynthesis. The resulting audio is intelligible most of the time. The last task that PAULE can fulfill is planning cp-trajectories for a given target acoustics and a target semantics, by jointly matching the target acoustics as closely as possible while at the same time predicting the target semantics. In all three tasks, the articulatory effort is kept minimal by enforcing stationarity (stay at the same position, if possible) and constant force (if the articulator position has to be changed, do it with constant force).

In order to plan cp-trajectories, PAULE utilises a semantic and an acoustic error, which is not calculated from an actual synthesis but from an imagined prediction of an expected outcome of the execution of the cp-trajectories. Figure 7.1 shows this fundamental principle. Planning on a predicted effect allows for an adaptive control model that can utilize local error information to plan and adjust the control dependent on the individual state of the VTL, while anticipating future demands in the semantic and acoustic goal space.

Additional evaluations with PAULE are needed but it is clear that PAULE can indeed model fine-grained anticipatory coarticulation structures, that it can condition on the current and past state of the VTL, and that it produces articulatory movements that are similar to articulator movements recorded in humans. Additionally, PAULE can be used to synthesize speech with the VTL by simply giving a target acoustics without the need of transcribing and aligning the speech in the audio on the word or segment level. Even if PAULE occasionally fails to synthesize intelligible speech, overall, PAULE is robust and can resynthesize speech that it has never encountered before and which it intrinsically cannot fully mimic. An example is the resynthesis of female speakers with the male vocal tract geometry of the VTL, which always leads to different formants in the resynthesized

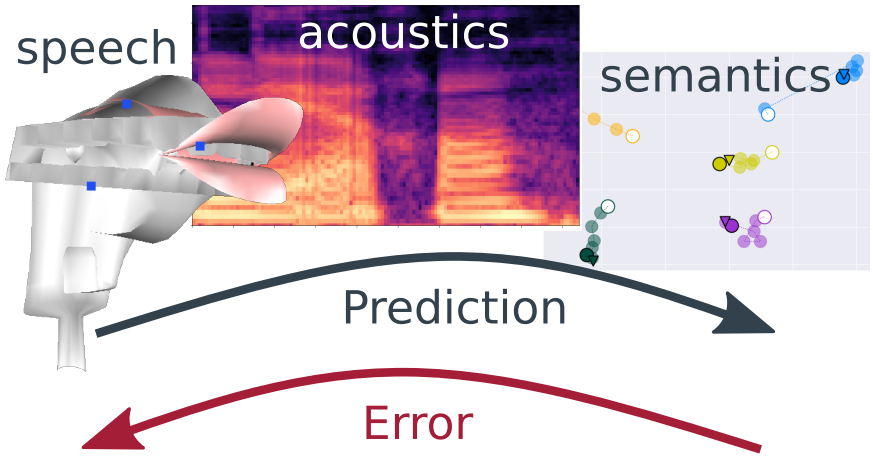

Figure 7.1: The main idea of PAULE is to predict the acoustics and semantics of the upcoming articulatory movements (cp-trajectories) and use the error between the intended target acoustics and the intended target semantics to plan and correct the movement.

produced audio compared to the target audio. This is possible due to the interleaved planning through the internal loop and the alignment of the predictive forward model through the outer loop. For now, PAULE is limited to small audio samples of half a second to a second of audio containing usually a single word.

7.1 Guiding principles

In the first chapter, guiding principles were formulated that inspired the design of PAULE in the following way:

PAULE works on the word level and takes the word as the smallest meaningful unit in speech. It makes it possible to initiate the process of speech synthesis with a semantic representation and plans trajectories in a goal-directed way to match a target meaning. The planning uses local gradients and is adaptive and error-driven. Furthermore, the planning is predictive or anticipatory in the sense that PAULE predicts the effect of the motor commands into a time window of around a second into the future. This prospective anticipation of its own behavior is then used to adapt and correct the behavior within this time frame.

On a longer time frame over multiple word utterances, PAULE absorbs experience into its predictive components. Furthermore, all components (or individual models) of PAULE are individually trained on training data from different collections of learning events, and no language-specific theoretical constructs such as phonemes or gestural scores are used. All components implement mappings from time-series to time-series, or from time-series to fixed-vectors. Therefore, PAULE is a computational model of speech production that learns incrementally.

The design of PAULE and its components is kept as simple as possible, while still achieving good synthesis results. The individual models are trained on relatively small speech corpora, which contain less than 20 hours of speech in total. Still it achieves moderate to good performance on out-of-vocabulary words, where the planning relies primarily on the acoustic error. Regarding all of these aspects, PAULE fulfills the guiding principles defined in the introduction. Note that PAULE is a proof of concept, that is deliberately kept simple and is exposed to limited training data. Even if it is probable, it is unknown if PAULE will scale up to the full vocabulary of a language and can be trained according to the experiences humans make during language learning.

7.2 Comparison DIVA and FACTS

Planning along the local gradients of a predictive forward model is a rather new approach to control. In contrast, in the DIVA model[110], an influential model of human speech production, a direct feedforward controller is learned and is correcting the cp-trajectories by feedforward feedback controllers that learn the mapping of the actual error between the current state of the vocal tract and the desired target state. Whereas PAULE implements an indirect method that utilize the gradients of the predictive forward model and plans on imagined effects, DIVA learns an explicit error correction from the actual error of the outer loop. Note that the feedforward controller in the DIVA model learns the same mapping as the direct inverse model in PAULE and suffers from the same one-to-many problem, which is solved by applying the error correction in the auditory and somatosensory space. Furthermore, the DIVA model is implemented on a substantially simpler articulatory speech synthesis model. Adapting DIVA to work and control the VTL would allow for a direct comparison between DIVA and PAULE.

A third modern control model is given by the FACTS model[69] that combines a state estimate of a predictive forward model with the error actually encountered in the articulatory synthesizer. FACTS utilizes a predictive error but in contrast to PAULE directly combines it with the actual error. Furthermore, in contrast to PAULE, the FACTS model has only a very short time horizon and only works on a simpler articulatory speech synthesizer. The authors of the FACTS model showed interest in porting FACTS to control the VTL, which would allow for a direct comparison to PAULE.

The last class of control models that exists, is developed by the group around Peter Birkholz at the TU Dresden, who created and maintains the VTL. They developed two classes of control models. One class is similar to the direct inverse model in PAULE. The other class optimizes the blending parameters of the gestural scores in a segment-based approach. This is similar to the deterministic segment-based approach used to generate the training data in PAULE, but adds an optimization or fitting procedure on top of the segment-based synthesis. In this optimization, the cp-trajectories are not changed directly, but the boundaries and relaxation parameters, as well as dominance values for the individual segments, are adjusted to better match the target acoustics[39].

All these control models have in common that they do not operate in the semantic space as a goal space and do not use any semantic error. They only circulate error between

the acoustic, motor, and somatosensory domain and therefore assume that the meaning transfer can purely be modeled in a feedforward manner. This is in strong contrast to PAULE, where the error in the expected meaning has a direct situational influence on the motor program.

7.3 Future work

Where DIVA implements error feedback not only from the outer loop, but also from the somatosensory feedback, PAULE relies on the error originating from the acoustic and semantic domains. Adding an error channel for the somatosensory feedback to PAULE is done in subsequent work[92]. The idea here is to use another predictive forward model that does not predict the acoustics but a low dimensional frontal state of the VTL oral cavity. This is achieved by extracting the minimal area function for each one-centimeter interval starting from the lips and going eight centimeters into the oral cavity. The reasoning behind this is to create an awareness and expectation in PAULE on how close the articulators are to each other and to generate knowledge of where constrictions are occurring. From this somatosensory constriction space, further mappings are learned to the acoustic representation and the semantic representation. The semantic and acoustic error is calculated in the same way as for the other pathways (see Chapter 5). Then the error is pushed back along the gradients of the different models, collected for all pathways, and summed.

In the current setup, PAULE makes substantial use of the outer loop. In the outer loop, PAULE mumbles and listens to itself 240 times for a single planning. This is necessary to align the predictive forward model with the physical simulation of the acoustics in the VTL. This 240 mumblings are only a few compared to a grid search or compared to how many are needed for an evolutionary algorithm. Nevertheless, they are too many to make PAULE fully behavioral plausible. One reason, making so many outer loop iterations necessary, is the distribution of the training data for the predictive forward model. The predictive forward model is trained on a segment-based synthesis approach, but the planning with its internal loop iterations changes the cp-trajectories into a different time dynamics. Therefore, the initial training data is only of limited use, the more the dynamics of the cp-trajectories is different to the one in the segment-based approach. To reduce the amount of mumblings, in a next step, training data will be generated with PAULE using the current amount of mumbling. This generated training data matches the time dynamics better and should therefore have better distributional properties for pre-training. A predictive forward model trained on this new training data hopefully needs a smaller number of mumblings, while still keeping the synthesis quality at the same level. In addition, a simulation study should investigate, how the effect of practice is on the number of mumblings and outer loop iterations. The number needed should dependent on the experience PAULE has in producing a word type. The outcome of this simulation study is that PAULE needs little to no mumblings and outer loop iterations for word types it knows well, but needs the current number of mumblings and iterations for a new word type.

Another important next step is to implement a method to produce and plan longer phrases like whole utterances or sentences. Besides longer computation times, planning longer utterances needs a design decision on the semantic representation level. One possibility would be to define one semantic vector per word in the utterance and define points in time where the predicted and produced semantic vector should be close to the target vector. The embedder should predict the semantic vector from the log-mel-spectrogram, starting after the previous word was ended until the current word is finished. Another option would be to collapse the full phrase onto a single semantic vector and require that the semantics is achieved at the end of the utterance.

Either way, the duration of the planning interval should be around 2.7 seconds and chunks of these 2.7 seconds should be planned with some overlap. This could be achieved by planning 2.7 seconds into the future while executing one word at a time. This should result in coarticulation patterns that take 2.7 seconds into account on the planned dynamics but could potentially be even longer ranging if the coarticulation depends on the initial state only as we reward stationary articulatory movements with the velocity loss.

Regarding the planning loss or planning error, it is important to point out that the weighting of the additive loss is crucial to achieving good planning results. For example, if the jerk loss dominates the additive loss, the planning will not change the curves substantially, as the initialization usually produces curves that have a low jerk and, therefore, are at a local minimum regarding the jerk loss landscape. If the velocity loss dominates the planning error this results in flat curves that look unnatural. A dominating acoustic loss removes the effects of the semantic components and might lead to highly oscillating cp-trajectories, as the regularizing effect of the velocity and jerk loss is not present. A dynamic weighting scheme might be useful and different adaptive loss schemes have been tested, but in the end having a constant weighting together with the ADAM optimizer yielded the most robust and best results. Losses are weighted so that acoustic and velocity losses are on the same order of magnitude and the semantic and jerk loss are one magnitude (ten times) smaller compared to the acoustic loss. This results in enough changes introduced by the planning and usually in a decent improvement in the acoustic and semantic loss in the finally produced word. Furthermore, the effect of an acceleration loss was investigated. The acceleration loss leads as expected to constant velocities and, therefore, constant shifts in the position of the articulators. These constant velocities are not present in human recordings of articulator movements. The acceleration loss therefore is not useful for the task of controlling a vocal tract lab synthesizer.

A positional loss might be interesting and will be further investigated after PAULE is able to synthesize longer utterances. A positional loss component might be introduced as a bias towards a resting, neutral, or default gesture. This might be especially important for longer utterances as they should start and end from a reasonable starting and ending position. The planning loss implemented in the current version of PAULE keeps the articulators in the last position as long as there is no need for a change through the acoustic or semantic prediction error. To bring the articulators into a resting position, which humans do, is out of the scope of the current implementation. It might be easily

implemented by attracting the cp-trajectories to a neutral gesture with an additional loss component.

At the moment the individual models within PAULE are pretrained on a German speech corpus. Since PAULE relies on the experience of the German sound repertoire, PAULE plans locally optimal cp-trajectories that depent on this German learning history. It will be an interesting next step to check how the synthesis quality for a second language like English or Mandarin is. Especially, the errors or speech variants that produced by PAULE are of interest and if these are the typical variants shown by German speakers acquiring the target second language. In order to test this quantitatively robust and reliable, effects of second language acquisition have to be identified and target speech recordings without the typical variants need to be resynthesized, by a PAULE model that is pretrained on German. If the, for second language learners, typical variants surface, this would show that the mechanisms implemented in PAULE can account for these effects.

This will yield only preliminary results though as second language acquisition should also be modelled by changing the embedder, which is the acoustic to semantics embedding. Changing the embedder can simulate the difficulties to discriminate all the relevant and meaningful sound patterns and acoustic differences in the new language, which occurs especially in the beginning. Lastly, the semantic embedding space is different in different languages and it remains unclear how to update the embedding space and respectively the embedder during the learning of the new language.

Still, it should be kept in mind that this does not mean that PAULE is implementing the speech production process as humans do it. We designed PAULE after guiding principles that are present in humans as well, but a lot more validation and testing are needed before any conclusion could be made about how well PAULE simulates human behavior. The decision to remove all symbol-like representations from the data pipelines in PAULE might be too strong. In future implementations, it might be useful to introduce a dynamic and hierarchical way of encoding often used movement patterns in an efficient coding scheme and retreiving and mixing them during speech production. The current implementation of the PAULE model without any gesture or phone representation can function as a reference to control models that rely on symbol-like structures in speech production. Such a comparision can give insights into how humans use symbolic representations during speech production and which effect can be attributed to the symbolic and combinatorial nature that is present in written western language systems and which aspects of speech production do not need any symbolic representation.

As PAULE is pretrained on data generated with the segment-based approach, another next step is to remove the training data from the process, at least for the predictive forward model and the direct inverse model and the GAN models. All these models can, in principle, be solely trained on the utterances produced by PAULE. In contrast, the embedder needs some external training data to first establish the semantic embedding space. This new model, called PAULEzero, would be trained in a goal babbling regime, where it tries to explore the semantic and acoustic goal space. Goal babbling has successfully been implemented for vowels and simple consonant-vowel syllables[57,71]

with the VTL. For the semantics, even a simplified goal space could be used at the beginning, which only distinguishes between an affirmative and a rejective utterance. PAULEzero would therefore implement a simplified version of how babies learn to talk. The pretraining of the embedder is the ability of the children to already understand many utterances of their language and the goal babbling and simplified semantic space resemble the babbling phase and the interactions between the caregiver and the baby.

Besides changing the training regime with PAULEzero, the individual models and the model architecture can be updated to more modern concepts like Transformer-based networks. This PAULEtransformer can be compared to the current PAULE implementation, which is based on LSTM-layers, as well as to a PAULEmlp implementation, where all individual models are implemented using simple MLP models (see Section 5.5.2). Together with a deterministic benchmark like `articubench`, this allows to quantify the differences according to different scores in the articulatory speech synthesis domain. Alternatively to the Transformer-based predictive forward model, a complex-valued ANN could be implemented. This complex-valued ANN could emulate a fast Fourier transform and could emulate the transformation from the time domain to the frequency domain.

Additionally, for fully automatic benchmarks like `articubench`, a human evaluation of the synthesis quality of different control models for the VTL is important to further validate the control models. This human evaluation experiment should be conducted after full phrases can be synthesized and the somatosensory feedback is added to PAULE. The main reasoning is that the spliced-out words that are used as target right now are often difficult to understand even for the human recordings. This is a well-known fact, when words are taken out of the context of the utterance they are uttered in[4]. If the identical acoustics is presented in the context of the utterance the perceived acoustics changes and it is substantially easier for the listener to understand and identify the word.

7.4 Conclusion

In conclusion, PAULE allows to model articulation starting from semantics by using a predictive framework. Within this framework optimal movement trajectories are planned by minimizing an error in an acoustic and semantic target space. The planning of optimal articulations depends on the experiences that PAULE already has with its articulations. Furthermore, the planning relies on two kinds of loops: an internal loop, which only imagines the acoustics; and an outer loop, where PAULE listens to itself.

Glossary

acoustic representation The acoustic representation defines the data format how acoustics is stored and therefore which perceptual sensations are encoded. In PAULE a logarithmized Mel banks spectrogram (log-mel-spectrogram) is used for the acoustic representation. The acoustic representation can be instantiated as a target acoustics, predicted acoustics, or produced acoustics. A detailed description of the acoustic representation can be be found in Section 3.2. 10, 24, 40, 45–47, 54, 64–71, 76, 87, 88, 94, 96, 141, 146–148

acoustic-only task The acoustic-only task, also known as copy-synthesis or mimicking task, is the task to find a proper motor program for an articulatory speech synthesizer that produces the same or a very similar acoustics to a target acoustics. In this thesis the acoustic-only task requires to find suitable cp-trajectories for the VTL to minimize an error to a given target acoustics. In the acoustic-only tasks a target semantics is not given. The two tasks that allow for a target semantics are the semantic-only task and the semantic-acoustic task. The three different tasks are describe in detail in the `articubench` benchmark in Section 6.6. 118, 126, 130–132, 148, 149

articulatory speech synthesizer An articulatory speech synthesizer is a computer program that simulates in different approximations the human speech system. The articulatory speech synthesizer takes movement commands as inputs and outputs synthesized speech as an audio signal. To get from the movement commands to the audio signal the articulatory speech synthesizer models the continuous movement and generates air pressure changes through a physical acoustical simulation. The articulatory speech synthesizer used in this thesis is the VocalTractLab (VTL), which takes cp-trajectories as inputs and is described in detail in Chapter 2. 33, 145–149

control model In the scope of this thesis a *control model* finds suitable inputs to an articulatory speech synthesizer. It therefore plans and controls the movement of the articulators like the human brain plans and controls the movements of the human articulators like the tongue and the jaw. The control model introduced in this thesis is PAULE, which is described in detail in Chapter 5. 33, 147

cp-GAN The Wasserstein-GAN trained to sample control parameter trajectories (cp-trajectories) (cp-GAN) is part of PAULE and maps the semantic representation to the cp-trajectories. It is used to initialize the cp-trajectories before the first internal

loop iteration starts in the semantic-only task. It is a fixed-vector to time-series model and is described in detail in Section 5.4.4. 145

cp-trajectories The control parameter trajectories (cp-trajectories) are the inputs to the VTL, which is the articulatory speech synthesizer used in this thesis. They define the movements of the 30 cps every 110 samples of a 44,100 Hz audio signal, i. e. approximately every 2.5 milliseconds. The cp-trajectories are described in detail in Section 3.1. 5, 19, 26, 37, 40, 42, 145–148

direct inverse model The direct inverse model is part of PAULE and maps the acoustic representation to the cp-trajectories. It is used to initialize the cp-trajectories before the first internal loop iteration starts in the acoustic-only (copy-synthesis) and the semantic-acoustic task. It is a time-series to time-series model and is described in detail in Section 5.4.3. 73–75, 78, 80, 98–101, 103, 104, 118, 120, 121, 125, 134, 139, 140, 143

embedder The embedder is part of PAULE and maps the acoustic representation to the semantic representation. It is used in the iterations of the internal loop and in the iterations of the outer loop. It is a time-series to fixed-vector model and is described in detail in Section 5.4.2. 67, 68, 71–73, 76, 78, 87, 91, 94–98, 105, 118, 119, 121, 122, 130, 131, 134, 141–143, 147, 148

gestural score A gestural score is a definition of articulatory movement trajectories or targets for the articulatory movements for a specific phone or syllable. A sequence of gestural scores of the VTL can be orchestrated and blended together to form cp-trajectories in a segment-based approach. This segment-based approach is described in Section 4.1. 38

human speech system The human speech system consists of the lungs, the glottis, the oral and nasal cavities, the jaw, the tongue, and the lips. A short description on how the human speech system produces speech is given in Section 1.2. 5, 17, 21–23, 25, 27–29, 145, 149

internal loop The internal loop of PAULE is the circle of starting with a target semantics, predicting from some initial cp-trajectories an acoustic representation and the semantic representation and finally comparing these predicted acoustics and predicted semantics with the target acoustics and target semantics. The internal loop is described in detail in Section 5.2.2. 20, 21, 49, 52, 69–76, 78, 79, 87, 89, 94, 96, 98, 103, 138, 144, 146–148

LSTM Long-Short-Term-Memory (LSTM) is a certain type of artificial neural network cell or layer that allows to model long ranging variable length sequence modeling. In the standard PAULE implementation LSTM-layers are the dominant layer type that does most of the work. 7, 80, 82, 84, 147

mel-GAN The Wasserstein-GAN trained to sample log-mel-spectrograms (mel-GAN) is part of PAULE and maps the semantic representation to the acoustic representation. It is used to initialize the acoustic representation before the first internal loop iteration starts in the semantic-only task. It is a fixed-vector to time-series model and is described in detail in Section 5.4.4. 147

outer loop The outer loop of PAULE is the circle of starting with a target semantics, finding the cp-trajectories to this target semantics, synthesizing the cp-trajectories with VTL, embedding the synthesized word into the semantic space and comparing the produced semantics of the synthesized word with the target semantics. The outer loop is described in detail in Section 5.2.3. 5, 20, 49, 66, 68–70, 76, 79, 91, 94, 96, 138, 139, 144, 146

PAULE The Predictive Articulatory speech synthesis Utilizing Lexical Embeddings (PAULE) framework is a control model for the VTL articulatory speech synthesizer. It is the main topic of this thesis and described in detail in Chapter 5. 3, 5, 7, 11, 17, 33, 63, 75, 137, 147

predicted acoustics The predicted acoustics is the output of the predictive forward model and can be interpreted as an imagined acoustical representation on how given control parameter trajectories (cp-trajectories) would sound like. In contrast to the produced acoustics the predicted acoustics is not based on the synthesis of the VTL synthesizer. The data structure of the predicted acoustics is a logarithmized Mel banks spectrogram (log-mel-spectrogram). The predicted acoustics is described in detail in Section 3.2.2. 45, 47, 49, 52, 69, 71, 73, 77, 78, 87, 94, 118, 145–147

predicted semantics The predicted semantics is the output of the embedder, which gets the predicted acoustics as input. In contrast to the produced semantics the predicted semantics is not based on the synthesis of the VTL synthesizer. The data structure of the predicted semantics is a fastText semantic vector. The predicted semantics is described in detail in Section 3.3.2. 45, 49, 52, 69, 71, 73, 78, 94, 118, 119, 146, 148

predictive forward model The predictive forward model is part of PAULE and maps the cp-trajectories to the acoustic representation. This mapping bypasses the articulatory speech synthesizer and allows is the defining element of the internal loop. The predictive forward model is a time-series to time-series model. It is described in detail in Section 5.4.1. 5, 49, 69–73, 76, 78, 79, 87–94, 96, 98, 100, 105, 118, 134, 138–140, 143, 147

produced acoustics The produced acoustics is derived from the synthesized audio, which is the result of the speech synthesis of the VocalTractLab (VTL) synthesizer. The VTL has control parameter trajectories (cp-trajectories) as inputs and the audio signal as outputs. The data structure of the predicted acoustics is a logarithmized Mel banks spectrogram (log-mel-spectrogram). The produced acoustics is described

in detail in Section 3.2.3. 45, 47, 49, 52, 66, 76, 77, 79, 94, 96, 118, 119, 121, 122, 132, 145, 147, 148

produced semantics The produced semantics is the output of the embedder, which gets the produced acoustics as input. Therefore, it is a semantics that results from a synthesis of the VTL synthesizer. The data structure of the produced semantics is a fastText semantic vector. The produced semantics is described in detail in Section 3.3.3. 45, 52, 68, 76, 79, 94, 98, 118, 119, 130, 132, 147, 148

semantic representation The semantic representation defines the data format how meaning is stored and therefore which internal meaning structure is encoded. For the semantic representation a semantic lexical word embedding vector (semantic vector) from the fastText project is used. The semantic representation can be instantiated as a target semantics, predicted semantics, or produced semantics. A detailed description of the semantics representation can be be found in Section 3.3. 27, 31, 45, 50, 54, 67, 69, 71, 96, 138, 141, 146–149

semantic-acoustic task The semantic-acoustic task, also known as supervised generation task, is the task to find a proper motor program for an articulatory speech synthesizer that produces the same or a very similar acoustics to a target acoustics. Additionally, to the high similarity in the acoustic representation a high similarity or identity in the semantic representation is required. Therefore, the semantic-acoustic task involves a pre-specified target semantics as well as a target acoustics. In this thesis the semantic-acoustic task requires to find suitable cp-trajectories for the VTL to minimize an error to a given target semantics and target acoustics. The two other tasks that are defined in this thesis are the semantic-only task, which only requires to match a target semantics, and an acoustic-only task, which only requires to match a target acoustics. The three different tasks are describe in detail in the `articubench` benchmark in Section 6.6. 118, 130–134, 145, 149

semantic-only task The semantic-only task, also known as full generation task, is the task to find a proper motor program for an articulatory speech synthesizer that produces the same or a very similar semantics to a target semantics. In this thesis the semantic-only task requires to find suitable cp-trajectories for the VTL to minimize an error to a given target semantics. As the semantic representation does not encode any duration of the produced word, the semantic-only task requires to pre-define a duration along the definition of the target semantics. The two other tasks that are defined in this thesis are the semantic-acoustic task, which requires to match a target semantics and a target acoustics simultaneously, and an acoustic-only task, which only requires to only match a target acoustics. The three different tasks are describe in detail in the `articubench` benchmark in Section 6.6. 118, 119, 122, 130–133, 145, 148

target acoustics The target acoustics defines the acoustical target in the perceptual space that should be reached with a synthesis. Usually there is a target semantics

alongside the target acoustics. The target acoustics is described in detail in Section 3.2.1. 20, 22, 27, 32, 40, 45, 47, 49, 51, 52, 63–66, 68, 69, 71–78, 87, 94, 98, 103, 104, 113, 117–122, 130, 137, 138, 140, 145, 146, 148, 149

target semantics The target semantics defines a semantic target in the meaning space that should be reached with a synthesis. Usually there is a target acoustics alongside the target semantics. The target semantics is described in detail in Section 3.3.1. 22, 27, 40, 45, 49, 51, 52, 63–65, 67–69, 71–73, 75–78, 94, 98, 104, 113, 118, 119, 130, 133, 137, 138, 145–149

VTL The VocalTractLab (VTL) is a geometrical, computational model of the human speech system that allows to move the articulators and allows to synthesize a speech signal as a mono audio file. The VTL is the specific implementation of an articulatory speech synthesizer used for PAULE. The VTL is described in detail in Chapter 2. 3, 5, 8, 10, 30, 35, 37, 40, 64, 75, 145, 148, 149

Abbreviations

cp-GAN Wasserstein-GAN trained to sample cp-trajectories

cp-trajectories control parameter trajectories

LSTM Long-Short-Term-Memory

mel-GAN Wasserstein-GAN trained to sample log-mel-spectrograms

PAULE Predictive Articulatory speech synthesis Utilizing Lexical Embeddings

VTL VocalTractLab

`articubench` ARTICUlatory speech synthesis BENCHmark

Adam ADAptive Moment estimation

ANN Artificial Neural Network

CMA-ES Covariance Matrix Adaptation Evolution Strategy

Common Voice Mozilla Common Voice corpus

cp control parameter

DFKI Deutsches Forschungszentrum für Künstliche Intelligenz

DLR Deutsches Zentrum für Luft- und Raumfahrt

EMA electromagnetic articulography

ESSV Konferenz zur elektronischen Sprachsignalverarbeitung

fastText fastText Library for efficient text classification and representation learning

FIAS Frankfurt Institute for Advanced Studies

FOSS Free and Open Source Software

GAN Generative Adversarial Network

GECO IMS GECO database

ISSP International Seminar on Speech Production

KEC Karl Eberhard Corpus

LDL Linear Discriminative Learning

log-mel-spectrogram logarithmized Mel banks spectrogram

MFA Montreal Forced Aligner

MLP Multi-Layer-Perceptron

MPI Max-Planck-Institut

ReLU Rectified Linear Unit

RMSE Root Mean Squared Error

RNN Recurrent artificial Neural Network

semantic vector semantic lexical word embedding vector

SGD Stochastic Gradient Decent

TeaP Tagung experimentell arbeitender Psychologen

UMAP Uniform Manifold Approximation and Projection for Dimension Reduction

Bibliography

[1] Ardila, R., Branson, M., Davis, K., Henretty, M., Kohler, M., Meyer, J., Morais, R., Saunders, L., Tyers, F. M., and Weber, G. (2019). Common voice: A massively-multilingual speech corpus. *CoRR*, abs/1912.06670. Cited on pages 53 and 58.

[2] Arlinger, S. (1990). Manual of practical audiometry. *Ear and Hearing*, 11(3):244. Cited on page 31.

[3] Arnold, D., Lopez, F., Sering, K., Tomaschek, F., and Baayen, H. (2016). Acoustic speech learning without phonemes: Identifying words is olated from spontaneous speech as a validation for a discriminative learning model for acoustic speech learning. Talk, TeaP. Cited on page 10.

[4] Arnold, D., Tomaschek, F., Sering, K., Lopez, F., and Baayen, R. H. (2017). Words from spontaneous conversational speech can be recognized with human-like accuracy by an error-driven learning algorithm that discriminates between meanings straight from smart acoustic features, bypassing the phoneme as recognition unit. *PloS one*, 12(4):e0174623. Cited on pages 10, 18, 46, 116, and 137.

[5] Arppe, A., Hendrix, P., Milin, P., Baayen, R. H., Sering, K., and Shaoul, C. (2013). ndl: Naive discriminative learning. Cited on page 9.

[6] Baayen, R. H. (2001). *Word Frequency Distributions*. Kluwer Academic Publishers, Dordrecht. Cited on page 56.

[7] Baayen, R. H., Chuang, Y.-Y., Shafaei-Bajestan, E., and Blevins, J. P. (2019). The discriminative lexicon: A unified computational model for the lexicon and lexical processing in comprehension and production grounded not in (de) composition but in linear discriminative learning. *Complexity*, 2019. Cited on pages 11 and 20.

[8] Beaman, K. V. and Sering, K. (2022). Measuring change in lectal coherence across real-and apparent-time. In *The Coherence of Linguistic Communities*, pages 87–105. Routledge. Cited on page 11.

[9] Beaman, K. V. and Tomaschek, F. (2021). Loss of Historical Phonetic Contrast across the Lifespan: Articulatory, Lexical, and Social Effects on Sound Change in Swabian. In *Language Variation and Language Change Across the Lifespan*, pages 209–234. Routledge. Cited on page 16.

[10] Beaman, K. V., Tomaschek, F., and Sering, K. (2021). The cognitive coherence of sociolects across the lifespan: A case study of swabian german. Cited on page 11.

[11] Berg, K. (2013). Graphemic alternations in English as a reflex of morphological structure. *Morphology*, 23(4):387–408. Publisher: Springer. Cited on page 16.

[12] Berg, K. and Aronoff, M. (2017). Self-organization in the spelling of english suffixes: The emergence of culture out of anarchy. *Language*, 93(1):37–64. Cited on page 16.

[13] Berk, L. E. (2020). *Entwicklungspsychologie*, volume 4334. Pearson Deutschland GmbH. Cited on pages 15, 18, 19, and 21.

[14] Berthier, N. E. (1996). Learning to reach: a mathematical model. *Developmental psychology*, 32(5):811. Cited on page 21.

[15] Birkholz, P. (2013). Modeling consonant-vowel coarticulation for articulatory speech synthesis. *PloS one*, 8(4):e60603. Cited on pages 3, 5, 10, 35, and 37.

[16] Bloomfield, L. (1983). An introduction to the study of language. *An Introduction to the Study of Language*, pages 1–383. Cited on page 15.

[17] Boersma, P. and Weenink, D. (2022). Praat: doing phonetics by computer. http://www.praat.org. Cited on page 56.

[18] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. Cited on page 16.

[19] Buddeberg, K. and Grotlüschen, A. (2020). *LEO 2018: Leben mit geringer Literalität*. wbv. Cited on page 15.

[20] Butz, M. V., Bilkey, D., Humaidan, D., Knott, A., and Otte, S. (2019a). Learning, planning, and control in a monolithic neural event inference architecture. *Neural Networks*, 117:135–144. Cited on page 20.

[21] Butz, M. V., Bilkey, D., Humaidan, D., Knott, A., and Otte, S. (2019b). Learning, planning, and control in a monolithic neural event inference architecture. *Neural Networks*, 117:135–144. Cited on pages 20 and 73.

[22] Chafe, W. and Tannen, D. (1987). The relation between written and spoken language. *Annual review of anthropology*, 16(1):383–407. Cited on page 15.

[23] Chomsky, N. and Halle, M. (1968). *The sound pattern of English*. Harper and Row, New York. Cited on page 15.

[24] Coker, C. H. (1976). A model of articulatory dynamics and control. *Proceedings of the IEEE*, 64(4):452–460. Cited on page 20.

[25] De Saussure, F. (1966). *Course in General Linguistics*. McGraw, New York. Cited on pages 16 and 17.

[26] Denes, P. (1955). Effect of Duration on the Perception of Voicing. *The Journal of the Acoustical Society of America*, 27(4):761–764. Cited on page 16.

[27] Developers, T. (2022). Tensorflow. See the full list of authors https://github.com/tensorflow/tensorflow/graphs/contributors. Cited on page 78.

[28] Ernestus, M. (2000). *Voice assimilation and segment reduction in casual Dutch. A corpus-based study of the phonology-phonetics interface*. LOT, Utrecht. Cited on page 56.

[29] Fels, S., Vogt, F., Van Den Doel, K., Lloyd, J., Stavness, I., and Vatikiotis-Bateson, E. (2006). Artisynth: A biomechanical simulation platform for the vocal tract and upper airway. In *International Seminar on Speech Production, Ubatuba, Brazil*, volume 138. Cited on page 35.

[30] Fink, J., Widmann, A., Sering, K., and Exner, C. (2016). Attentional bias triggers disgust-specific habituation problems in subclinical contamination-based obsessive-compulsive disorder. Poster, TeaP. Cited on page 10.

[31] Fink-Lamotte, J., Widmann, A., Sering, K., Schröger, E., and Exner, C. (2021). Attentional processing of disgust and fear and its relationship with contamination-based obsessive–compulsive symptoms: Stronger response urgency to disgusting stimuli in disgust-prone individuals. *Frontiers in psychiatry*, 12. Cited on page 11.

[32] Fischer, S. R. (2003). *History of writing*. Reaktion books. Cited on page 15.

[33] Flash, T. and Hogan, N. (1985). The coordination of arm movements: an experimentally confirmed mathematical model. *Journal of neuroscience*, 5(7):1688–1703. Cited on page 21.

[34] Fleischer, M., Mainka, A., Kürbis, S., and Birkholz, P. (2018). How to precisely measure the volume velocity transfer function of physical vocal tract models by external excitation. *PLOS ONE*, 13(3):1–16. Cited on pages 35 and 36.

[35] Fowler, C. A. and Turvey, M. T. (1980). Immediate Compensation in Bite-Block Speech. *Phonetica*, 37(5-6):306–326. Publisher: Karger Publishers. Cited on page 21.

[36] French, N. R. and Steinberg, J. C. (1947). Factors governing the intelligibility of speech sounds. *The journal of the Acoustical society of America*, 19(1):90–119. Cited on page 39.

[37] Gahl, S. (2008). Time and thyme are not homophones: The effect of lemma frequency on word durations in spontaneous speech. *Language*, 84(3):474–496. Cited on pages 23 and 56.

[38] Gao, Y. (2021). Articulatory copy synthesis based on the speech synthesizer vocaltractlab. Cited on page 37.

[39] Gao, Y., Stone, S., and Birkholz, P. (2019). Articulatory copy synthesis based on a genetic algorithm. In *INTERSPEECH*, pages 3770–3774. Cited on page 133.

[40] Grave, E., Bojanowski, P., Gupta, P., Joulin, A., and Mikolov, T. (2018). Learning word vectors for 157 languages. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*. Cited on pages 19 and 49.

[41] Greenberg, S. (1999). Speaking in shorthand - A syllable-centric perspective for understanding pronunciation variation. *Speech Communication*, 29:159–176. Cited on page 16.

[42] Grotlüschen, A., Buddeberg, K., Dutz, G., Heilmann, L. M., and Stammer, C. (2021). Leo 2018 - living with low literacy (public use file). GESIS Data Archive, Cologne. ZA6266 Data file Version 1.0.0, https://doi.org/10.4232/1.13771. Cited on page 15.

[43] Hanson, H. M. and Stevens, K. N. (2002). A quasiarticulatory approach to controlling acoustic source parameters in a klatt-type formant synthesizer using hlsyn. *The Journal of the Acoustical Society of America*, 112(3):1158–1182. Cited on page 35.

[44] Harley, T. A. (2013). *The psychology of language: From data to theory*. Psychology press. Cited on pages 15 and 18.

[45] Hasselhorn, M. and Schneider, W. (2007). *Handbuch der Entwicklungspsychologie*. Hogrefe Verlag GmbH & Company KG. Cited on pages 15, 18, 19, and 21.

[46] Hay, J. B., Pierrehumbert, J. B., Walker, A. J., and LaShell, P. (2015). Tracking word frequency effects through 130 years of sound change. *Cognition*, 139:83–91. Publisher: Elsevier. Cited on page 16.

[47] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9:1735–80. Cited on page 81.

[48] Iskarous, K., Goldstein, L., Whalen, D. H., Tiede, M., and Rubin, P. (2003). Casy: The haskins configurable articulatory synthesizer. In *International Congress of Phonetic Sciences, Barcelona, Spain*, pages 185–188. Cited on page 35.

[49] Johnson, K. (2004a). Massive reduction in conversational American English. In *Spontaneous speech: data and analysis. Proceedings of the 1st session of the 10th international symposium*, pages 29–54, Tokyo, Japan. The National International Institute for Japanese Language. Cited on page 16.

[50] Johnson, K. (2004b). Massive reduction in conversational American English. In *Spontaneous speech: data and analysis. Proceedings of the 1st session of the 10th international symposium*, pages 29–54, Tokyo, Japan. The National International Institute for Japanese Language. Cited on page 24.

[51] Joo, J., Steen, F. F., and Turner, M. (2017). Red hen lab: Dataset and tools for multimodal human communication research. *KI-Künstliche Intelligenz*, 31(4):357–361. Cited on pages 16 and 58.

[52] Kemps, R., Ernestus, M., Schreuder, R., and Baayen, H. (2004). Processing reduced word forms: The suffix restoration effect. *Brain and Language*, 90(1-3):117–127. Cited on page 23.

[53] Keune, K., Ernestus, M., Van Hout, R., and Baayen, R. H. (2005a). Social, geographical, and register variation in Dutch: From written 'mogelijk' to spoken 'mok'. *Corpus Linguistics and Linguistic Theory*, 1:183–223. Cited on page 24.

[54] Keune, K., Ernestus, M., Van Hout, R., and Baayen, R. H. (2005b). Social, geographical, and register variation in Dutch: From written 'mogelijk' to spoken 'mok'. *Corpus Linguistics and Linguistic Theory*, 1:183–223. Cited on page 56.

[55] Kingma, D. P. and Ba, J. L. (2015). Adam: A method for stochastic optimization. *3rd International Conference for Learning Representations*, abs/1412.6980. Cited on pages 75 and 76.

[56] Klimaj, Z. and Makarski, W. (1969). Językoznawstwo w katolickim uniwersytecie lubelskim. *Roczniki Humanistyczne*, 17(1):79–99. Cited on page 22.

[57] Krug, P. K., Birkholz, P., Gerazov, B., van Niekerk, D. R., Xu, A., and Xu, Y. (2022). Efficient exploration of articulatory dimensions. *Studientexte zur Sprachkommunikation: Elektronische Sprachsignalverarbeitung*, 2022:51–58. Cited on page 136.

[58] Landauer, T. and Dumais, S. (1997). A Solution to Plato's Problem: The Latent Semantic Analysis Theory of Acquisition, Induction and Representation of Knowledge. *Psychological Review*, 104(2):211–240. Cited on page 15.

[59] Linke, M. and Ramscar, M. (2020). How the Probabilistic Structure of Grammatical Context Shapes Speech. *Entropy*, 22(1):90. Publisher: Multidisciplinary Digital Publishing Institute. Cited on page 16.

[60] Loeb, G. E., Brown, I. E., and Cheng, E. J. (1999). A hierarchical foundation for models of sensorimotor control. *Experimental brain research*, 126:1–18. Cited on page 21.

[61] Lohmann, A. (2018). Time and thyme are not homophones: A closer look at Gahl's work on the lemma-frequency effect, including a reanalysis. *Language*, 94(2):e180–e190. Cited on page 23.

[62] Maeda, S. (1990). Compensatory articulation during speech: Evidence from the analysis and synthesis of vocal-tract shapes using an articulatory model. In *Speech production and speech modelling*, pages 131–149. Springer. Cited on page 35.

[63] McAuliffe, M., Socolof, M., Mihuc, S., Wagner, M., and Sonderegger, M. (2017). Montreal forced aligner: Trainable text-speech alignment using kaldi. In *INTERSPEECH*. Cited on page 59.

[64] McInnes, L., Healy, J., and Melville, J. (2020). Umap: Uniform manifold approximation and projection for dimension reduction. Cited on page 92.

[65] Murakami, M., Kröger, B., Birkholz, P., and Triesch, J. (2015). Seeing [u] aids vocal learning: Babbling and imitation of vowels using a 3d vocal tract model, reinforcement learning, and reservoir computing. In *2015 joint IEEE international conference on development and learning and epigenetic robotics (ICDL-EpiRob)*, pages 208–213. IEEE. Cited on page 27.

[66] Nam, H., Goldstein, L., Saltzman, E., and Byrd, D. (2004). Tada: An enhanced, portable task dynamics model in matlab. *The Journal of the Acoustical Society of America*, 115(5):2430–2430. Cited on pages 15 and 35.

[67] Norris, D. and McQueen, J. M. (2008). Shortlist B: a Bayesian model of continuous speech recognition. *Psychological review*, 115(2):357. Cited on page 23.

[68] Otte, S., Schmitt, T., Friston, K., and Butz, M. V. (2017). Inferring adaptive goal-directed behavior within recurrent neural networks. In *International Conference on Artificial Neural Networks*, pages 227–235. Springer. Cited on page 20.

[69] Parrell, B., Ramanarayanan, V., Nagarajan, S., and Houde, J. (2019). The FACTS model of speech motor control: Fusing state estimation and task-based control. *PLoS computational biology*, 15(9):e1007321. Cited on pages 17, 35, and 133.

[70] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc. Cited on page 78.

[71] Philippsen, A. (2021). Goal-directed exploration for learning vowels and syllables: a computational model of speech acquisition. *KI-Künstliche Intelligenz*, 35(1):53–70. Cited on page 136.

[72] Philippsen, A. K. (2018). *Learning How to Speak. Goal Space Exploration for Articulatory Skill Acquisition*. PhD thesis, Bielefeld: Universität Bielefeld. Cited on pages 20 and 27.

[73] Philippsen, A. K., Reinhart, R. F., and Wrede, B. (2016). Goal babbling of acoustic-articulatory models with adaptive exploration noise. In *2016 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pages 72–78. IEEE. Cited on page 27.

[74] Plag, I. (2018). *Word-formation in English*. Cambridge University Press. Cited on page 15.

[75] Port, R. F. and Leary, A. P. (2005). Against formal phonology. *Language*, 81:927–964. Cited on page 24.

[76] Reetz, H. and Jongman, A. (2011). *Phonetics: Transcription, Production, Acoustics, and Perception*. John Wiley & Sons. Cited on page 16.

[77] Saltzman, E. and Byrd, D. (2000). Task-dynamics of gestural timing: Phase windows and multifrequency rhythms. *Human Movement Science*, 19(4):499–526. Cited on page 20.

[78] Sapir, E. (1921). An introduction to the study of speech. *Language*, 1. Cited on page 15.

[79] Schmager, P. (2017). http://www.vocaltractlab.de/index.php?page=targetoptimizer-about. Cited on page 56.

[80] Schmidt-Barbo, P. (2021). Using semantic embeddings to start and plan articulatory speech synthesis. Cited on page 100.

[81] Schmidt-Barbo, P., Otte, S., Butz, M. V., Baayen, R. H., and Sering, K. (2022). Using semantic embeddings for initiating and planning articulatory speech synthesis. *Studientexte zur Sprachkommunikation: Elektronische Sprachsignalverarbeitung 2022*, pages 32–42. Cited on page 11.

[82] Schmidt-Barbo, P., Shafaei-Bajestan, E., and Sering, K. (2021). Predictive articulatory speech synthesis with semantic discrimination. *Studientexte zur Sprachkommunikation: Elektronische Sprachsignalverarbeitung 2021*, pages 177–184. Cited on page 11.

[83] Schneider, W. and Lindenberger, U. (2018). *Entwicklungspsychologie*. Beltz. Cited on pages 15, 18, 19, and 21.

[84] Schweitzer, A. and Lewandowski, N. (2013). Convergence of articulation rate in spontaneous speech. In *INTERSPEECH*, pages 525–529. Cited on page 56.

[85] Sering, K. (2016). Listen and speak: Training a vocal tract synthesis model using natural speech. first ideas and problems. Alberta Phonetics Laboratory, Edminton, Canada. Cited on page 10.

[86] Sering, K. (2019a). Learning vocal tract control parameters to synthesize speech. Neural Information Processing Group, Tübingen, Germany. Cited on page 11.

[87] Sering, K. (2019b). Learning vocal tract control parameters to synthesize speech. MoProc Workshop, Tübingen, Germany. Cited on page 11.

[88] Sering, K. (2021). Predictive articulatory speech synthesis utilizing lexical embeddings (paule). Spoken Morphology Colloquium, Düsseldorf, Germany. Cited on page 11.

[89] Sering, K. (2022). Predictive articulatory speech synthesis utilizing lexical embeddings (paule). Neural Information Processing Group, Tübingen, Germany. Cited on page 11.

[90] Sering, K., Milin, P., and Baayen, R. H. (2018). Language comprehension as a multi-label classification problem. *Statistica Neerlandica*, 72(3):339–353. Cited on pages 11, 19, and 49.

[91] Sering, K. and Schmidt-Barbo, P. (2022). Articubench - An articulatory speech synthesis benchmark. *Studientexte zur Sprachkommunikation: Elektronische Sprachsignalverarbeitung 2022*, pages 43–50. Cited on pages 11, 61, and 125.

[92] Sering, K. and Schmidt-Barbo, P. (2023). Somatosensory feedback in PAULE. *Studientexte zur Sprachkommunikation: Elektronische Sprachsignalverarbeitung 2023*, pages 119–126. Cited on pages 11 and 134.

[93] Sering, K., Schmidt-Barbo, P., Otte, S., Butz, M. V., and Baayen, H. (2020). Recurrent gradient-based motor inference for speech resynthesis with a vocal tract simulator. In *12th International Seminar on Speech Production*. Cited on pages 11, 86, and 94.

[94] Sering, K., Stehwien, N., Gao, Y., Butz, M. V., and Baayen, H. (2019). Resynthesizing the geco speech corpus with vocaltractlab. *Studientexte zur Sprachkommunikation: Elektronische Sprachsignalverarbeitung 2019*, pages 95–102. Cited on pages 11, 45, and 53.

[95] Sering, K. and Tomaschek, F. (2020). Comparing kec recordings with resynthesized ema data. *Studientexte zur Sprachkommunikation: Elektronische Sprachsignalverarbeitung 2020*, pages 77–84. Cited on pages 11, 45, 60, 86, 94, and 124.

[96] Sering, K., Tomaschek, F., and Saito, M. (2021). Anticipatory coarticulation in predictive articulatory speech modeling. *Studientexte zur Sprachkommunikation: Elektronische Sprachsignalverarbeitung 2021*, pages 208–215. Cited on pages 11, 61, 121, 122, and 123.

[97] Sering, K., Weitz, M., Künstle, D.-E., and Schneider, L. (2017). Pyndl: Naive discriminative learning in python. Cited on page 9.

[98] Shafaei-Bajestan, E., Moradipour-Tari, M., Uhrig, P., and Baayen, R. H. (2021). Ldl-auris: a computational model, grounded in error-driven learning, for the comprehension of single spoken words. *Language, Cognition and Neuroscience*, 0(0):1–28. Cited on page 51.

[99] Shahmohammadi, H., Heitmeier, M., Shafaei-Bajestan, E., Lensch, H., and Baayen, H. (2022). Language with vision: a study on grounded word and sentence embeddings. *arXiv preprint arXiv:2206.08823*. Cited on page 49.

[100] Shaoul, C., Schilling, N., Bitschnau, S., Arppe, A., Hendrix, P., Sering, K., and Baayen, R. H. (2015). Ndl2: Naïve discriminative learning. Cited on page 9.

[101] Shcherba, L. (1993). Russian vowels in qualitative and quantitative respects. *Journal of Voice*, 7(3):270–290. Cited on page 22.

[102] Steiner, I., Tomaschek, F., Bolkart, T., Hewer, A., Wuhrer, S., and Sering, K. (2018). Head and tongue model: Simultaneous dynamic 3d face scanning and articulography. Simphon.net Meeting, Stuttgart, Germany. Cited on page 11.

[103] Stevens, S. S., Volkmann, J., and Newman, E. B. (1937). A scale for the measurement of the psychological magnitude pitch. *The journal of the acoustical society of america*, 8(3):185–190. Cited on page 31.

[104] Story, B. H. (2011). Tubetalker: An airway modulation model of human sound production. In *Proceedings of the First International Workshop on Performative Speech and Singing Synthesis*, pages 1–8. P3S 2011, Vancouver, Canada. Cited on pages 15, 33, and 35.

[105] Strunk, J., Schiel, F., Seifart, F., et al. (2014). Untrained forced alignment of transcriptions and audio for language documentation corpora using webmaus. In *LREC*, pages 3940–3947. Cited on page 16.

[106] Tomaschek, F., Arnold, D., Broeker, F., and Baayen, R. H. (2018a). Lexical frequency co-determines the speed-curvature relation in articulation. *Journal of Phonetics*, 68:103–116. Cited on page 22.

[107] Tomaschek, F., Arnold, D., Sering, K., and Strauss, F. (2021). A corpus of schlieren photography of speech production: potential methodology to study aerodynamics of labial, nasal and vocalic processes. *Language Resources and Evaluation*, 55(4):1127–1140. Cited on pages 10 and 11.

[108] Tomaschek, F., Arnold, D., Sering, K., Tucker, B. V., van Rij, J., and Ramscar, M. (2020). Articulatory variability is reduced by repetition and predictability. *Language and speech*. Cited on pages 11 and 22.

[109] Tomaschek, F., Tucker, B. V., Fasiolo, M., and Baayen, R. H. (2018b). Practice makes perfect: the consequences of lexical proficiency for articulation. *Linguistics Vanguard*, 4(s2):1–13. Cited on page 22.

[110] Tourville, J. A. and Guenther, F. H. (2011). The DIVA model: A neural theory of speech acquisition and production. *Language and cognitive processes*, 26(7):952–981. Cited on pages 15, 17, 20, 35, and 133.

[111] Trubetskoy, N. S. (1939). *Grundzüge der phonologie*. Van den Hoeck & Ruprecht. Cited on page 22.

[112] Tucker, B. V., Brenner, D., Danielson, D. K., Kelley, M. C., Nenadić, F., and Sims, M. (2019). The massive auditory lexical decision (MALD) database. *Behavior research methods*, 51(3):1187–1204. Cited on page 16.

[113] Umesh, S., Cohen, L., and Nelson, D. (1999). Fitting the mel scale. In *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No. 99CH36258)*, volume 1, pages 217–220. IEEE. Cited on page 31.

[114] van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. Cited on pages 15, 24, and 35.

[115] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30. Cited on page 83.

[116] Wagner, P., Trouvain, J., and Zimmerer, F. (2015). In defense of stylistic diversity in speech research. *Journal of Phonetics*, 48:1–12. Cited on page 16.

[117] Wang, Y., Skerry-Ryan, R., Stanton, D., Wu, Y., Weiss, R. J., Jaitly, N., Yang, Z., Xiao, Y., Chen, Z., Bengio, S., Le, Q., Agiomyrgiannakis, Y., Clark, R., and Saurous, R. A. (2017). Tacotron: Towards end-to-end speech synthesis. Cited on pages 24 and 35.

[118] Zipf, G. K. (1935). *The Psycho-Biology of Language*. Houghton Mifflin, Boston. Cited on page 53.