Generalizing speech recognition techniques for monotonic sequence modeling applications



Brendan Shillingford

Somerville College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Michaelmas 2019

Acknowledgements

I am indebted to my supervisors Nando de Freitas and Phil Blunsom for countless hours of advice, guidance, and support in every possible way. I would also like to thank my examiners, Professor Kyunghyun Cho and Professor Thomas Lukasiewicz, for their helpful criticism and thorough feedback which improved the quality of the final version of this thesis.

I would like to acknowledge the Clarendon Fund for its funding of my studies as a Clarendon Scholar, and DeepMind for an internship where I learned plenty of what I know now.

We would like to thank NVIDIA for its generous donations of several Titan X and Titan Xp GPUs and a DGX-1 used in work throughout this thesis. Parts of the work in this thesis were supported by funding from EPSRC and CIFAR. I would like to thank Áine Jackson, Brittany Klug and Samantha Pugh for helping us measure the experienced lipreader baseline in Chapter 3, along with Mitko Sabev for his phonetics guidance, Odysseas Votsis for his video production help, and Alex Graves and Oiwi Parker Jones for many helpful comments. I would like to thank Mahalo to M. Puakea Nogelmeier for providing Oiwi and I with an electronic copy of *Hi'iakaikapoliopele* (Ho'oulumāhiehie, 2007), indispensable for our work in Chapter 2. For help with Chapter 4 and Chapter 5, I would like to thank Hagen Soltau for preparation of the YouTube audio dataset that our dataset is based on, and for providing the language model used for decoding. For help regarding Chapter 4 I would also like to thank Andrew Zisserman for his valuable contributions as an advisor; Shane Agnew for his assistance; and Misha Denil, Sean Legassick, Iason Gabriel, Dominic King, and Alan Karthikesalingam for their helpful comments.

I would like to thank Misha Denil, Nando de Freitas, and Oiwi Parker Jones for their valuable feedback throughout this thesis.

I would like to thank my friends and labmates in Oxford, Jakob, Marcin, Misha, Oana, Oiwi, Yannis, and Ziyu, who made my time at Oxford truly memorable. I would like to thank Quill, as she has really made my time in London extra special. Finally, last but certainly not least, I would like to thank my family for their support and always encouraging me to learn and explore.

Abstract

This thesis demonstrates how modeling techniques from speech recognition can be advantageous in a variety of other problems. The problems to which we apply these techniques include visual-only speech recognition (lipreading), restoration of characters missing from old Hawaiian orthography but present in the modern orthography, and an interactive lipreading-based keyboard.

In particular, modeling techniques from speech recognition carry two main benefits applied throughout this thesis. The first is their exploitation of monotonicity, a property found widely across sequence modeling problems besides just speech. The second is the separation of the problem into a combination of learned and fixed pieces, the latter of which facilitates the injection of expert knowledge (in the form of finite state transducers) at arbitrary places in a larger system. These features for encoding inductive biases about our problem improve data efficiency, which is especially beneficial when paired data for learning the end-to-end problem from scratch is insufficiently plentiful or entirely unavailable. We can thus get the best of both worlds: explicit representation of our expert knowledge and other inductive biases, and learned neural network components for those that are hard to model by hand and have plentiful data.

Contents

List of Abbreviations **xi**

1 Introduction **1**
- 1.1 Thesis structure and contributions . . . 4
- 1.2 Brief overview of speech recognition . . . 7
  - 1.2.1 Acoustic model . . . 8
  - 1.2.2 Hidden Markov models . . . 9
  - 1.2.3 Language models . . . 11
  - 1.2.4 Modern developments . . . 12
- 1.3 Review of modelling sequences with neural networks . . . 14
  - 1.3.1 Recurrent neural networks . . . 14
  - 1.3.2 Spatiotemporal convolutions . . . 18
  - 1.3.3 Modeling sequences with neural networks . . . 19
- 1.4 Brief introduction to finite state transducers . . . 21
  - 1.4.1 Finite state acceptors . . . 21
  - 1.4.2 Weighted FSAs . . . 24
  - 1.4.3 Example: n-gram language models . . . 26
  - 1.4.4 Weighted finite state transducers . . . 26
  - 1.4.5 Example: speech recognition . . . 29
  - 1.4.6 Example: modernizing Hawaiian orthography . . . 31
- 1.5 Connectionist temporal classification . . . 32
  - 1.5.1 Probability of a particular sequence . . . 33
  - 1.5.2 Sampling CTC outputs . . . 35
  - 1.5.3 Prediction . . . 36
  - 1.5.4 Decoding with CTC and FSTs . . . 37
  - 1.5.5 Conclusions . . . 39

2 Restoring Hawaiian orthography using FSTs and a hybrid FST/RNN approach **41**
- 2.1 Introduction . . . 41
- 2.2 Related work . . . 43
- 2.3 Data . . . 43

2.3.1 Phonemes & the modern orthography . . . 43
2.3.2 Modern & missionary orthographies . . . 44
2.3.3 Data sources . . . 45
2.4 Models . . . 45
2.4.1 End-to-end FSTs . . . 46
2.4.2 Hybrid approach: Composing FSTs and RNNLMs . . . 47
2.5 Results . . . 49
2.5.1 Summary of language models trained . . . 49
2.5.2 Evaluation . . . 50
2.5.3 Simulated parallel corpus (modern texts) . . . 51
2.5.4 Real parallel corpus (newspaper texts) . . . 51
2.5.5 Predictions on newspaper excerpts . . . 52
2.6 Conclusions and future work . . . 53

3 LipNet: visual speech recognition using connectionist temporal classification 55
3.1 Introduction . . . 56
3.2 Related Work . . . 57
3.3 LipNet . . . 60
3.3.1 Gated Recurrent Unit . . . 60
3.3.2 Connectionist Temporal Classification . . . 61
3.3.3 LipNet Architecture . . . 61
3.3.4 Baseline-LSTM Architecture . . . 62
3.3.5 Implementation . . . 63
3.4 Lipreading Evaluation . . . 63
3.4.1 Data Augmentation . . . 63
3.4.2 Baselines . . . 64
3.4.3 Performance Evaluation . . . 64
3.4.4 Learned representations . . . 66
3.5 Conclusions . . . 69

4 Large-scale visual speech recognition 71
4.1 Introduction and motivation . . . 71
4.2 Potential medical applications . . . 73
4.3 Related work . . . 76
4.4 A data pipeline for large-scale visual speech recognition . . . 78
4.5 An efficient spatiotemporal model of visual speech recognition . . . 83
4.6 Evaluation . . . 87
4.6.1 Results . . . 89
4.6.2 Phoneme confusions . . . 90

4.6.3 Phoneme deletions, insertions, substitutions 92
4.6.4 Face rotation vs. performance heatmap 92
4.6.5 Saliency maps 92
4.6.6 Transfer evaluation on LRS3-TED 93
4.7 Conclusions 95

5 Interactive decoding of words from visual speech recognition models 97
5.1 Introduction 97
5.2 Related work 98
5.3 Method 99
5.4 Evaluation 102
5.5 Conclusions 104

6 Conclusions 105

Appendices

A Sample predictions for Chapter 2 125

List of Abbreviations

ASR . . . . . Automated Speech Recognition

CER . . . . . Character Error Rate, see also WER

CNN . . . . . Convolutional Neural Network

CTC . . . . . Connectionist Temporal Classification

DNN . . . . . Deep Neural Network (usually means feed-forward)

FSA . . . . . Finite State Acceptor

FST . . . . . Finite State Transducer

GRU . . . . . Gated Recurrent Unit (Recurrent Neural Network)

LM . . . . . Language Model

LSTM . . . . . Long Short-Term Memory (Recurrent Neural Network)

LSVSR . . . . Large-Scale Visual Speech Recognition

PER . . . . . Phoneme Error Rate, see also WER

RNNLM . . . Recurrent Neural Network Language Model

RNN . . . . . Recurrent Neural Network

WER . . . . . Word Error Rate, word-level edit distance of two strings divided by the ground-truth length

WFSA . . . . . Weighted Finite State Acceptor

WFST . . . . . Weighted Finite State Transducer ("weighted" is usually implied and omitted)

1 Introduction

This thesis attempts to answer the question: how can methods from speech recognition, such as classical symbolic methods and dynamic programming, be applied to other deep learning-based monotonic sequence transduction applications?

Speech recognition (or automated speech recognition, ASR) is the problem of recognizing text given an audio signal containing speech. In this work, we illustrate how techniques and modeling approaches that have proven useful in audio-based automatic ASR can be reapplied to several other problems, including visual speech recognition (lipreading), a Hawaiian orthography transduction problem, and interactively decoding from a lipreading model.

Perhaps the most notable feature of the ASR problem is monotonicity: the order in which words occur in a speech audio signal is the same order in which those words would appear in a transcript, barring small local movements (e.g. '$1' and 'one dollar'). While it is possible to add monotonicity constraints to standard deep learning models, the tools used specifically for ASR, such as finite state transducers (Mohri et al., 2002), connectionist temporal classification (Graves et al., 2006), and recurrent neural network transducers (Graves, 2012a), are specifically designed for the task and potentially more easily applicable. Most other work on sequence modeling in deep learning has been based on general sequence-to-sequence approaches without monotonicity, since they have their origins in machine translation where monotonicity does not usually apply (Sutskever et al., 2014; Cho et al., 2014).

For the problems described in this thesis, we avoid the use of encoder-decoder RNN approaches, as these force the model to learn from data that the output sequences are in the same order as the input sequence. Instead, we use methods

such as finite state transducers and connectionist temporal classification, which have monotonicity built-in.

Another feature common to many ASR models is that the problem of predicting words from audio is split into a combination of learned and fixed pieces. The fixed pieces facilitate the incorporation of structure and expert knowledge at arbitrary places in the pipeline, complementing the learned pieces. Specifically, a classical ASR pipeline might split the problem as follows:

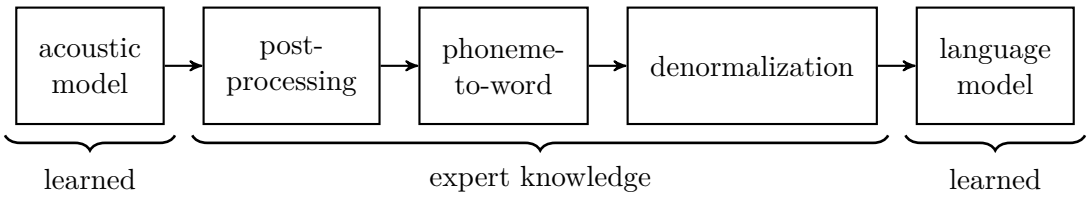


Figure 1.1: Components in a typical ASR pipeline and how they are connected.

As seen in the figure, common components include a pronunciation-to-word mapping (e.g. mapping from the phonemic representation /peɪmiː/ to the words 'pay me') and a normalization transformation (e.g. 'one dollar' ↔ '$1', or 'twenty twelve' ↔ {'2012', '20 12'}). Both of these are already well-understood problems; this gives a convenient way to encode our existing knowledge into an ASR pipeline. Learning them from scratch can be wasteful. An additional advantage is that the division into components allows us to take advantage of an abundance of data for a component at any point in the pipeline, such as text-only data for training the language model. Conveniently, this entire pipeline and its constituent pieces can be expressed in the formalism of finite state transducers (FSTs).

In general, we may have data for part of a problem, and the solution of another part can be easily expressed by hand or is otherwise already well-studied. This method gives us a way to add an inductive bias and our existing knowledge into a system satisfying these properties. In the ASR case, the pronunciation of words is well-known information so a phoneme to word mapping can be constructed explicitly, as can the process by which written language is translated to words of spoken language (text normalization).

Lipreading, or visual speech recognition, is the problem of decoding spoken text from a sequence of image frames showing a speaker's mouth. We follow a similar structure for lipreading, combining learned and fixed components into a larger system. In lipreading, we have the problem that much of the vocal tract is not visible, resulting in groups of different-sounding phonemes appearing the same visually. To address this, we can take advantage of the structural constraints that a

phoneme dictionary imposes on the outputs of the phoneme recognition component. Note also that explicitly modeling phonemes as an intermediate representation is beneficial compared to directly predicting characters or words, particularly for languages such as English where the relationship between orthography (a word's representation in written language) and pronunciation is non-trivial. For instance, in English, the 's' in the word 'debris' does not result in a consonant sound; other languages have mostly consistent mappings from orthography to pronunciation, or even a nearly bijective one (e.g. Korean).

We also formulate the translation of old Hawaiian orthography (which was missing many letters) to the modern orthography as a one-to-many mapping using FSTs, allowing us to encode our existing knowledge of how Hawaiian orthography has changed, producing a space of candidates that a language model searches through.

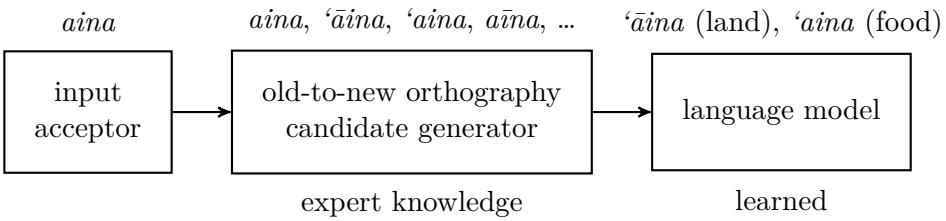


Figure 1.2: Predicting new Hawaiian orthography from old Hawaiian orthography, making use of a structured representation of expert knowledge. Example words are shown above: both *ʻāina* (land) and *ʻaina* (food) would appear ambiguously as aina in the old orthography, despite having different meanings and being pronounced differently.

Notice that the structures and expert knowledge that we are encoding above are all *discrete*. In end-to-end deep learning, where a single deep neural network is trained to directly perform a task from paired input-output data without intermediate supervision, it is often difficult to encode inductive biases and expert knowledge. It is easier to encode such structures for continuous data, but it remains an open problem for complex (large) discrete structures such as language, programs, or knowledge graphs, due to their nondifferentiability. When data is sufficiently plentiful, the advantage of encoding inductive biases, priors, or structures diminishes; indeed, Bottou and Gallinari (1991) suggest that learning an entire system together on the intended overall task metric, as opposed to separately on several sub-task metrics, results in higher performance.

The success of such an approach, however, relies on plentiful end-to-end data. Therefore, when we are in a regime where we have a 'moderate' amount of data, where the meaning of 'moderate' depends heavily on the task, the ability to incorporate expert knowledge and existing, known information becomes crucial. Lipreading, for

instance, is extremely data-hungry—despite the dataset we construct in Chapter 4 being an order of magnitude larger and more varied than existing ones, Makino et al. (2019) suggest that yet more data still helps. Similarly, ASR is increasingly trained end-to-end, but this is due to extremely large datasets produced by (semi-)automated methods (e.g. Lecouteux et al., 2012; Liao et al., 2013; Soltau et al., 2017); this is not feasible for domains in which labeled data is difficult to collect.

Being able to target inductive biases in the form of structure and expert knowledge, at arbitrary points in a system including sandwiched between learned components, is paramount in ASR for medium-sized datasets. We draw from these ideas in this thesis, applying them to problems with similar challenges.

1.1 Thesis structure and contributions

For some problems, we may have no paired data at all, such as the task of filling in missing characters in Hawaiian orthography explored in Chapter 2. In this case, the idea of combining an expert-designed system with a learned component (trained on a plentiful unpaired data source) is crucial.

Modern Hawaiian orthography includes characters for long vowels and glottal stops, but these are absent in older writing system of the 19th century, leading to old texts being ambiguous and difficult to read for modern readers. Given older Hawaiian text, our goal is to fill in these missing characters automatically. We construct an orthography-change transducer for encoding our prior knowledge (easily expressed as a finite state transducer), in combination with a learned language model, resulting in an FST pipeline similar to the decoding pipelines used for ASR. This removes the need for paired training data; without it, it is not at all obvious how to solve this problem with the limited corpora available for an endangered language like Hawaiian. This chapter also serves as an introductory example of using finite state transducers for modeling string transformations, as the approach looks quite similar to those in speech recognition. This chapter is based on the following papers:

> B. Shillingford* and O. Parker Jones* (2018). Recovering missing characters in old Hawaiian writing. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4929–4934.

> O. Parker Jones* and B. Shillingford* (2018). Composing RNNs and FSTs for Small Data: Recovering Missing Characters in Old Hawaiian Text. *Interpretability and Robustness in Audio, Speech, and Language Workshop at NeurIPS 2018.*

In Chapter 3, we make use of connectionist temporal classification and spatiotemporal (volumetric) convolutions to perform visual speech recognition (lipreading) at the sequence level, predicting sequences of graphemes (characters). Most previous work focused on the word level, which requires alignments.

> Y. Assael*, B. Shillingford*, S. Whiteson, and N. de Freitas (2016). LipNet: Sentence-level Lipreading. *arXiv preprint arXiv:1611.01599*.

In Chapter 4, we construct a large video dataset and construct a phoneme recognition model. Here, we build on the lipreading problem introduced in Chapter 3, and combine it with FST tools exemplified in Chapter 2. We separate the lipreading problem into one of phoneme recognition and one of decoding from phonemes, much like standard audio-based speech recognition. We argue that phonemes allow for better uncertainty modeling, which is crucial for lipreading. The model borrows some ideas from Chapter 3 such as spatiotemporal convolutions but also tackles problems unique to a large video processing model of this size.

> B. Shillingford*, Y. Assael*, M. W. Hoffman, T. Paine, C. Hughes, U. Prabhu, H. Liao, H. Sak, K. Rao, L. Bennett, M. Mulville, M. Denil, B. Coppin, B. Laurie, A. Senior, and N. de Freitas (2019). Large-scale visual speech recognition. *Proc. Interspeech 2019*, pp. 4135–4139.

> B. Shillingford*, Y. Assael*, M. W. Hoffman, T. Paine, C. Hughes, U. Prabhu, H. Liao, H. Sak, K. Rao, L. Bennett, M. Mulville, M. Denil, B. Coppin, B. Laurie, A. Senior, and N. de Freitas (2018). Large-scale visual speech recognition. *arXiv preprint arXiv:1807.05162*.

As the word error rate of lipreading models is too high for their predictions to be of direct use, at least with large open domain vocabularies, in Chapter 5 we introduce a novel method to address this. Suppose a human is trying to input a sentence into a mobile device using lipreading. As an alternative way to address the high phoneme recognition uncertainties observed in lipreading, we add interactivity into the decoder's search procedure, where a human helps guide the decoding process for an utterance that they themselves have spoken. We propose a 'human-in-the-loop' paradigm for a human to select the result of a phoneme recognition model word-by-word, and propose an algorithm for solving this problem.

> B. Shillingford, Y. Assael, and M. Denil (2019). Interactive decoding of words from visual speech recognition models. *In submission*.

*Note: * denotes equal contribution.

Other work

My following additional work is not included in this thesis:

T. Makino, H. Liao, Y. Assael, B. Shillingford, B. Garcia, O. Braga, and O. Siohan (2019). Recurrent Neural Network Transducer for Audio-Visual Speech Recognition. *Proceedings of IEEE Automatic Speech Recognition and Understanding Workshop.*

A. Gupta, B. Shillingford, Y. Assael, and T. C. Walters (2019). Speech Bandwidth Extension with Wavenet. *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pp. 205–208.

Y. Chen, Y. M. Assael, B. Shillingford, D. Budden, S. E. Reed, H. Zen, Q. Wang, L. C. Cobo, A. Trask, B. Laurie, Ç. Gülçehre, A. van den Oord, O. Vinyals, and N. de Freitas (2018). Sample Efficient Adaptive Text-to-Speech. *arXiv preprint arXiv:1809.10460.*

R. P. Costa, I. A. Assael, B. Shillingford, N. de Freitas, and T. Vogels (2017). Cortical microcircuits as gated-recurrent neural networks. *Advances in Neural Information Processing Systems*, pp. 272–283.

M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas (2016). Learning to learn by gradient descent by gradient descent. *Advances in Neural Information Processing Systems*, pp. 3981–3989.

1.2 Brief overview of speech recognition

This chapter briefly overviews how a typical speech recognition system may be constructed. Automatic speech recognition (ASR) is the problem of recognizing text given an audio signal containing speech. That is, given a representation of an audio signal $\mathbf{a}$, we want to find what sequence of words $\mathbf{w}$ is spoken in the audio:

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} p(\mathbf{w}|\mathbf{a}).$$

In the recent past, such a system was nearly invariably built using hidden Markov models (HMMs) to *generatively* represent audio. (We assume the reader is familiar with HMMs.) Traditional speech recognition systems model the relationship between audio and words using a noisy-channel approach, using Bayes rule to instead model

$$p(\mathbf{w}|\mathbf{a}) = \frac{p(\mathbf{a}|\mathbf{w})p(\mathbf{w})}{p(\mathbf{a})},$$

where $p(\mathbf{a}|\mathbf{w})$ is usually represented using an HMM. Since $p(\mathbf{a})$ does not depend on $\mathbf{w}$, we have

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} p(\mathbf{w}|\mathbf{a}) = \arg\max_{\mathbf{w}} p(\mathbf{a}|\mathbf{w})p(\mathbf{w}). \tag{1.2.0.1}$$

However, the audio produced for a given word is typically nontrivially related to the word—the speaker's voice, speed, pitch, pronunciation, etc. can vary significantly—generatively modeling $p(\mathbf{a}|\mathbf{w})$ directly is hard, especially in the era before deep generative models. Therefore, instead of conditioning on words, we condition on *phonemes*.

A phoneme is a discrete unit of speech that is considered perceptually distinct from others, in the context of a specific language such as English. For instance, /b/ is a consonant phoneme in English that occurs in various words such as "bat" or "bottle". Alternatively, one could think of a phoneme as a set of sounds in a language that are defined to be equivalent—such a set would include every way that the sound /b/, for example, could be pronounced and still be considered a /b/ sound. From this definition, the two words "bat" and "cat" are distinguished by /b/ versus /k/ (as written in the international phonetic alphabet (IPA)), thus /b/ and /k/ are both phonemes. In English, there are about 40 phonemes; the exact number depends on whether certain similar sounds are considered distinct.

Therefore, instead of directly modeling $p(\mathbf{a}|\mathbf{w})$, we introduce an intermediate phoneme sequence $\mathbf{p}$ and model

$$p(\mathbf{a}|\mathbf{w}) = \sum_{\mathbf{p}} p(\mathbf{a}|\mathbf{p})p(\mathbf{p}|\mathbf{w}).$$

The $\mathbf{w}$ to $\mathbf{p}$ mapping is many-to-many, in that multiple phoneme strings may map to the same word, and the same phoneme string may correspond to multiple words (homonyms). This mapping can either be modeled as a lookup table, where $p(\mathbf{p}|\mathbf{w})$ uniformly weighs each alternative pronunciation (phoneme sequence) for each word, or as a probability distribution, if we have data on which pronunciations are more common. The phoneme sequence to word mapping is called a *lexicon*. Phoneme labels can be extracted from an ASR corpus consisting of pairs of audio and word sequences (transcripts) using forced alignment: given an existing HMM model trained on the ASR corpus, and given some audio and associated transcripts (word sequences), we find the most likely HMM state sequence, given the audio and transcripts.[1] Sometimes, by making the assumption that one pronunciation is dominant, we simply approximate $p(\mathbf{a}|\mathbf{w}) = p(\mathbf{a}|\mathbf{p})$ where $\mathbf{p}$ is deterministically computed from $\mathbf{w}$ using a lookup table.

1.2.1 Acoustic model

The $\mathbf{a}|\mathbf{p}$ conditional distribution, called an *acoustic model* (AM), models audio features given linguistic features. Although audio features are important for ASR, we do not model audio in this thesis so we will not describe them in detail. Briefly, unprocessed one-dimensional waveforms are a possible feature representation, as are the more common frequency-based representations such as Mel-scaled spectrograms or Mel-frequency cepstral coefficients (MFCCs) (Zheng et al., 2001). One timestep of one of these representations, often computed over overlapping windows into the audio, is called a *frame*.

Other discrete labels derived from phonemes are sometimes used to take advantage of context, as the pronunciation of the same phoneme can vary significantly based on the phonemes surrounding it. One common alternative is triphones, a triple of a phoneme along with the phoneme preceding and proceeding it. As the number of triphones can be moderately large ((number of phonemes)^3, so for English approximately 40^3), one typically models context-dependent (CD) phonemes, where triphones are grouped into clusters that sound similar, instead of triphones directly. Clustering is typically performed using decision trees comprised of fixed questions that ask properties of the surrounding context, e.g. 'is the preceding phoneme a fricative?'. To contrast with CD phonemes, plain phonemes are often referred to as context-independent (CI) phonemes.

[1] In practice, one often assumes an ASR model is available for performing forced alignment, as the model used need not be as accurate as one used for ASR, since we are assuming we already know the text being spoken. However, if we do not have one, then expectation-maximization can be used to produce an improving sequence of HMMs: from a randomly or carefully initialized HMM, produce forced alignments, then train a new HMM, and iterate.

1.2.2 Hidden Markov models

In an HMM-based ASR system, the acoustic model is typically represented as an HMM with a particular sparse state connection structure. A common construction assigns three states per phoneme; let us assume this construction for concreteness. The state space of this HMM is $Z = \{z_{p,i} : \text{phoneme } p,\ i \in \{1,2,3\}\}$. The observation space of the HMM O is one of the audio representations mentioned above, such as MFCCs, time slices of a Mel-scaled spectrogram, or similar.

For a phoneme sequence $\{p_1, \ldots, p_T\}$ (where T is the number of audio frames), each phoneme has associated with it three states: $z_{p_i,1}$,$z_{p_i,2}$, and $z_{p_i,3}$. The states each have self-transitions and transitions to the next state, and next state transition for $z_{p_i,3}$ goes to $z_{p_{i+1},1}$. At the far ends, $z_{p_1,1}$ is the start state and $z_{p_T,3}$ is a final state with no transitions out of it. This state arrangement can be seen graphically in Figure 1.3:

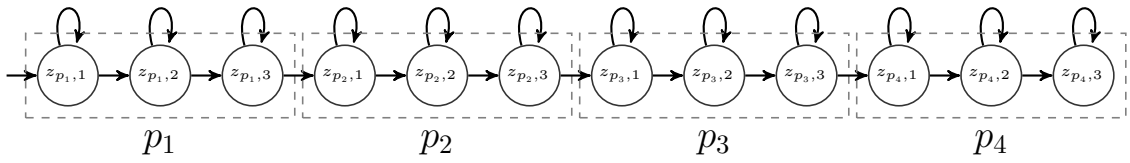


Figure 1.3: A example state connectivity structure for an HMM acoustic model with three states per phoneme, for a particular 4-phoneme sequence $\{p_1, \ldots, p_4\}$.

The state transition probabilities are associated with the identity of the phoneme p_i and the within-phoneme state index (1, 2, or 3), so that phonemes with the same identity share weights. This HMM is hierarchical, in the sense that given the sequence of phonemes $\mathbf{p}$, we can interpret the state structure as linking a separate HMMs for each phoneme deterministically, forming an HMM specific to this phoneme sequence (Gales, Young, et al., 2008).

The three states for each phoneme can be interpreted as the "beginning", "during", and "end" of the phoneme, three regions which are qualitatively different. This is a very minimal model, as how a given phoneme sounds can vary substantially, even after taking into account its rough position within the phoneme. If we additionally want to model the effects of preceding (or following) phonemes on the first (or third) states of the three states, then we can opt to use CD phonemes instead of CI phonemes for $\{p_1, \ldots, p_N\}$.

The emission probabilities $p(a_t|p_t)$ would usually be a Gaussian mixture model (GMM); an HMM model with GMM emission probabilities is called an HMM-GMM.

An example HMM with its transitions and observations for an example word 'bird' is illustrated in Figure 1.4.

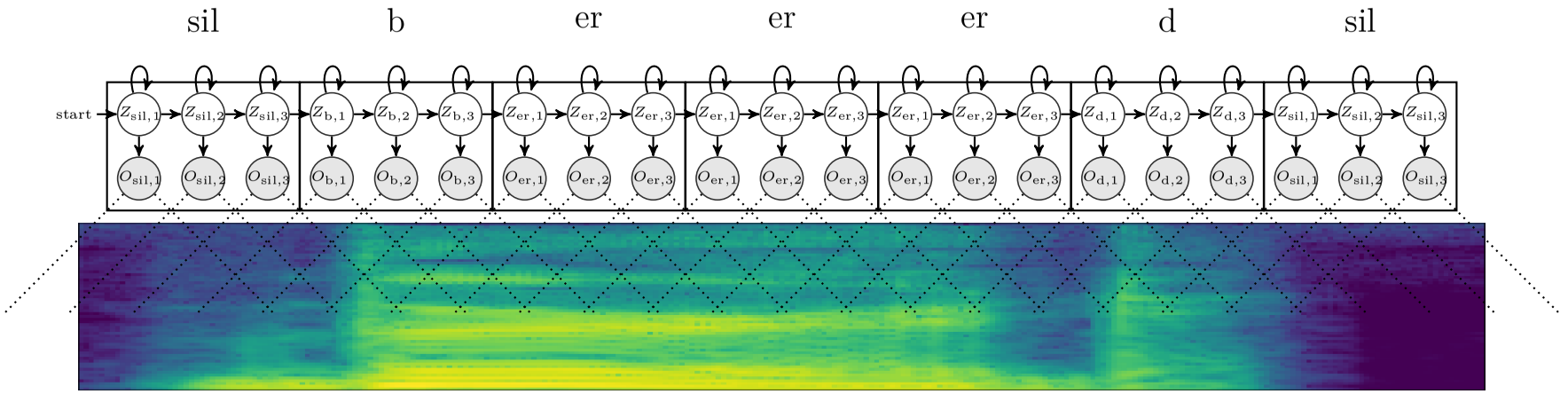


Figure 1.4: This figure shows a typical hidden Markov model-based acoustic model, computing the conditional probability of a Mel-scaled spectrogram for the word 'bird'. The HMM structure is constructed for a specific phoneme sequence by chaining together individual 3-state phoneme HMMs. Arrows between HMM hidden states z denote state transitions, and the arrows to the observation variables o denote emission connections which are typically modeled using Gaussian mixture models. In reality, the observations would typically be MFCCs, rather than a spectrogram, when using HMMs.

The HMM-GMM is then trained with expectation-maximization, with many tricks used for improving their performance, for example initializing the Gaussians in the GMM to the overall data statistics, and tying Gaussians together early in training followed by untying them later on.

Slightly more recently, the GMM emission distribution is replaced by a deep neural network, an approach called "hybrid" HMM-DNNs significantly surpassing the existing state-of-the-art in performance, held at the time by carefully-tuned HMM-GMMs. First, one trains a standard HMM-GMM and performs forced alignment. This produces an HMM state for each frame, which we treat as a label. We can think of these latent states learned by the HMM as finer-grained classes than phonemes, or perhaps as phonemes themselves if the HMM has few states. The deep neural network, being fed overlapping windows of audio as input, is then discriminatively trained to predict these HMM states. This is illustrated in Figure 1.5.

Roughly speaking, this models $p(\mathbf{p}|\mathbf{a})$ directly (where $\mathbf{p}$ is the HMM states rather than phonemes), which can be used in place of $p(\mathbf{a}|\mathbf{p})$ in the argmax with appropriate scaling (apply Bayes rule, then note that $p(\mathbf{a})$ is approximately constant if classes are balanced). See Hinton et al. (2012) for more details on this approach.

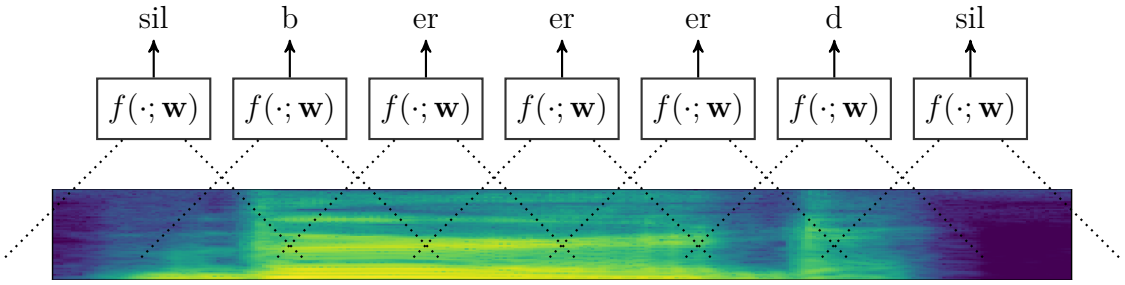
1.2.3 Language models

The other component of (1.2.0.1) is $p(\mathbf{w})$, modeling how likely a sequence of words is, called a *language model* (LM). This probability is broken down sequentially into

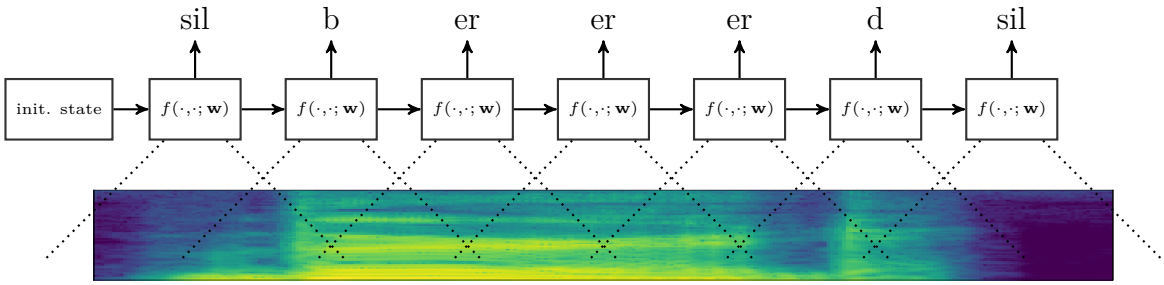
$$p(\mathbf{w}) = \prod_{i=1}^{W} p(w_i|w_1,\ldots,w_{i-1}),$$

each term of which can be modeled as conditioning on a fixed history by approximating $p(w_i|w_1,\ldots,w_{i-1}) \approx p(w_i|w_{i-n+1},\ldots,w_{i-1})$ for some fixed n, called n-gram models, or modeled autoregressively using recurrent neural networks to take advantage of all the conditioning history available. We assume the reader is already familiar with these.

In this section, we have seen all the components of a typical ASR system and how they are trained at a high level. At test time, all that is needed to be done is computation of the argmax in (1.2.0.1). As the sequence of words is discrete, this is solved using a variety of search algorithms, and is known as *decoding*. This can be quite model-dependent, as how to perform this search efficiently relies on properties of the model. For HMMs, we can take advantage of the Markov property, for instance. For phoneme-based modeling approaches, we can represent the lexicon ($p(\mathbf{w}|\mathbf{p})$) and language model ($p(\mathbf{w})$) together as a type of finite state



(a) A DNN-based discriminatively trained acoustic model. The arrow at the top of each replicated classifier box denotes the computation of a categorical likelihood (equivalently, a cross entropy loss).



(b) An RNN-based discriminatively trained acoustic model. Note the addition of a hidden state carried between timesteps, allowing the model to perform sequential modeling.

Figure 1.5: Two discriminatively-trained neural network-based acoustic models. Although phonemes are drawn above the classifier, typically HMM states would be used as targets, allowing for an easier drop-in replacement of the original HMM-based acoustic model. The targets are obtained from an existing HMM model using forced alignment.

automata called a finite state transducer, an approach we will see in Section 1.4. In this approach, given a distribution such as $p(\mathbf{p}|\mathbf{a})$ and the finite state transducer, decoding can be encapsulated into a search procedure.

1.2.4 Modern developments

We do not proceed into further detail on generative ASR models as the majority of modern approaches rely on discriminative training, modeling $p(\mathbf{w}|\mathbf{a})$ or $p(\mathbf{p}|\mathbf{a})$ directly. As is typical in machine learning, generative models perform well on moderate amounts of data, but larger amounts of data combined with discriminative models will often perform better; ASR is no exception.

Assuming we have per-frame phoneme labels for the audio signal, called *alignments*, we can train per-timestep classification models with any type of neural

network (recurrent, convolutional, or simply a feed-forward network that sees a window of audio). Modern systems that predict pre-frame phoneme labels still require a pretrained HMM-GMM for forced alignment, however, so methods that avoid the need for alignments are of interest. Later, in Section 1.5, we will see connectionist temporal classification, which marginalizes over alignments while still being trained discriminatively.

1.3 Review of modelling sequences with neural networks

This thesis assumes the reader is familiar with standard machine learning and deep learning tools and techniques. This includes basic probability, its application to machine learning learning, and optimization, such as maximum likelihood, stochastic gradient descent (SGD) and its variants such as Adam (Kingma and Ba, 2014), as well as basics of neural networks, such as multilayer perceptrons (feedforward neural networks), common activation functions (e.g. ReLU), and convolutional neural networks as typically used for tasks such as object recognition (e.g. AlexNet (Krizhevsky et al., 2012) and further advancements such as ResNet (He et al., 2016)). In this section, we briefly review common neural network components, besides those a typical reader can be assumed to already be familiar with. For a more general deep learning overview, see e.g. Goodfellow et al. (2016) (textbook), or LeCun et al. (2015) or Schmidhuber (2015) (review papers).

This section overviews recurrent neural networks (RNNs) and two common variants besides the Elman-style recurrent network formulation (Elman, 1990), long short-term memory RNNs (LSTMs) (Hochreiter and Schmidhuber, 1997) and gated recurrent units (GRUs) (Cho et al., 2014). This is followed by a description of how RNNs can be used to model conditional distributions of sequences. Several models described in this thesis take video as input and make important use of spatiotemporal convolutions, which process a volumetric (three-dimensional) signal. These are therefore also described and contrasted with other approaches for processing video with neural networks. An overview of connectionist temporal classification (CTC), which plays an important role for predicting discrete sequences without alignments, commonly used in speech recognition, is left for Section 1.5.

1.3.1 Recurrent neural networks

Recurrent neural networks are used to model sequences, or more specifically, model the transformation of an arbitrary-length sequence of vectors into another arbitrary-length sequence of vectors. In an RNN, this transformation is parameterized by a one-step neural network that maps from a *state* and an input to a new state and an output.

That is, given an input sequence $\{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$, an RNN models the output $\{\mathbf{z}_1, \ldots, \mathbf{z}_T\}$ as:

$$(\mathbf{h}_{i+1}, \mathbf{z}_{i+1}) = f(\mathbf{h}_i, \mathbf{x}_i; \boldsymbol{\theta}),$$

where $\mathbf{h}_i$ is the time-i *hidden state* (sometimes just called the state or RNN state), $\boldsymbol{\theta}$ is a collection of parameters, and $\mathbf{h}_0$ can be set to a learned value or the zero vector. That is, f maps $f : \mathbb{R}^{\text{state}} \times \mathbb{R}^{\text{input}} \to \mathbb{R}^{\text{state}} \times \mathbb{R}^{\text{output}}$. The input, state, and output sizes need not be the same as each other; for the sake of example, let us assume they are all in $\mathbb{R}^n$.

The RNN's *cell* function, f, parameterized by a set of weights as with any other neural network component, describes the computation of one timestep given the previous one. The length of the sequence need not be fixed in this setup.

The simplest possible RNN cell, the Elman-style RNN (Elman, 1990), chooses f as

$$\begin{aligned} \mathbf{h}_{i+1} &= g(\mathbf{W}_h \mathbf{h}_i + \mathbf{W}_x \mathbf{x}_i + \mathbf{b}) && (1.3.1.1) \\ \mathbf{z}_{i+1} &= \mathbf{h}_{i+1} && \text{state also used as output} \end{aligned}$$

where $\mathbf{W}_h$, $\mathbf{W}_x$, and $\mathbf{b}$ are parameters, g is an activation function such as sigmoid (where $\text{sigm}(r) = 1/(1 + \exp(-r))$) or tanh, and typically $\mathbf{h}_0 = \mathbf{0}$.

As with deep neural networks, the *vanishing gradient problem* occurs, leading to low-magnitude gradients at layers far from the loss function (Bengio et al., 1994; Hochreiter, 1998; Pascanu et al., 2013).

For an idea of why, consider an Elman network as shown in Equation (1.3.1.1), with its last output $\mathbf{h}_T$ composed with a scalar loss function L to give $L(\mathbf{h}_T)$. Then, differentiating Equation (1.3.1.1) yields

$$\frac{\partial \mathbf{h}_{i+1}}{\partial \mathbf{h}_i} = \text{diag}(g'(\cdots))\mathbf{W}^\top.$$

The gradient of the loss with respect to each timestep's hidden state is therefore given by

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{h}_t} &= \frac{\partial L}{\partial \mathbf{h}_T} \cdot \frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_{T-1}} \cdots \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t} \\ &= \frac{\partial L}{\partial \mathbf{h}_T} \cdot \text{diag}(g'(\cdots))\mathbf{W}^\top \cdots \text{diag}(g'(\cdots))\mathbf{W}^\top, \qquad (1.3.1.2) \end{aligned}$$

the norm of which is bounded as follows:

$$\left\| \frac{\partial L}{\partial \mathbf{h}_t} \right\| \leq \left\| \frac{\partial L}{\partial \mathbf{h}_T} \right\| \|\text{diag}(g'(\cdots))\| \left\|\mathbf{W}^\top\right\| \cdots \|\text{diag}(g'(\cdots))\| \left\|\mathbf{W}^\top\right\|.$$

Two classical choices of g for an Elman-style network, sigmoid or tanh, have derivatives with ranges (0,1/4] and $(0, 1]$ respectively. Therefore, we can simplify the bound to

$$\leq \left\| \frac{\partial L}{\partial \mathbf{h}_T} \right\| \left\|\mathbf{W}^\top\right\|^{T-t+1} \alpha^{T-t+1}$$

where α is the maximum of the absolute value of each entry of g' over all timesteps. Define $\alpha' = |\alpha \left\|\mathbf{W}^\top\right\||$, so:

$$\left\|\frac{\partial L}{\partial \mathbf{h}_t}\right\| \leq \left\|\frac{\partial L}{\partial \mathbf{h}_T}\right\| \alpha'^{T-t+1}. \tag{1.3.1.3}$$

If the absolute maximum eigenvalue (assuming diagonalizability) of $\mathbf{W}$ is less than 1, then $|\alpha'| < 1$ and the bound in Equation (1.3.1.3) decays to 0 exponentially in the number of timesteps of the RNN. This is known as the *vanishing gradient problem*, and impedes the learning of long-term dependencies.

A similar issue called *exploding* gradient exists as well, especially if g is chosen to be a function without a bounded derivative (e.g. ReLU). This is a lesser concern, however, as simple workarounds such as using a bounded activation function g or clipping of the g's output, or even clipping only the gradients, addresses this adequately. The former approach is taken in e.g. Hannun et al. (2014) when using a ReLU activation function. Gradient clipping is a common approach as suggested in Pascanu et al. (2013), and is commonly implemented by either ensuring the RNN's (or entire model's gradient's) gradient norm stays below a maximum, or by clipping the gradient element-wise.

The derivative of L with respect to the parameters, which is what we are actually interested in, e.g. $\mathbf{W}_h$, is trivially related to Equation (1.3.1.2):

$$\frac{dL}{d\mathbf{W}_h} = \frac{\partial L}{\partial \mathbf{h}_T} \sum_{t=1}^{T} \operatorname{diag}(g'(\mathbf{W}_h \mathbf{h}_{i-1} + \mathbf{W}_x \mathbf{x}_{i-1} + \mathbf{b})) \mathbf{W}_h^\top.$$

Also, a more realistic analysis where the loss function is applied to *every* output of the network instead of the just the last is presented in Pascanu et al. (2013).

Long short-term memory

To address this gradient propagation issue, alternatives to the classical Elman-style network such as the Long Short-Term Memory (LSTM) RNN cell have been developed (Hochreiter and Schmidhuber, 1997).

The LSTM cell with the forget gate extension of Gers et al. (2000) and without the "peephole" connection defines the RNN cell recurrence as:

$$\begin{aligned}
\mathbf{i}_t &= \operatorname{sigm}(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{b}_i) && \text{input gate} \\
\mathbf{f}_t &= \operatorname{sigm}(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{b}_f) && \text{forget gate} \\
\mathbf{o}_t &= \operatorname{sigm}(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{b}_o) && \text{output gate} \\
\mathbf{g}_t &= \tanh(\mathbf{W}_{xg}\mathbf{x}_t + \mathbf{W}_{hg}\mathbf{h}_{t-1} + \mathbf{b}_g) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t && \odot \text{ denotes element-wise product} \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t).
\end{aligned}$$

This is the LSTM formulation used in the work in this thesis.

Notably, the Jacobian of the full state $[\mathbf{h}_t, \mathbf{c}_t]$ with respect to that at the previous timestep, $[\mathbf{h}_{t-1}, \mathbf{c}_{t-1}]$, contains two paths: one through an activation function via $\mathbf{h}_t$ similar to the one in the Elman-style network, and an additive-only one via $\mathbf{c}_t$. The latter, disregarding the forget gate $\mathbf{f}_t$ for the moment, provides an additive path and therefore a gradient path without exponential decay of the gradient magnitude. By initializing $\mathbf{b}_f$ to $c\mathbf{1}$ for $c > 0$, we can make $\mathbf{f}_t$ close to $\mathbf{1}$ to reduce gradient vanishing at the start of training.

Gated recurrent unit

Gated Recurrent Unit (GRU) (Cho et al., 2014; Chung et al., 2014) is a type of recurrent neural network (RNN) that improves upon earlier RNNs by adding cells and gates for propagating information over more time-steps and learning to control this information flow. It is similar to the LSTM described in the previous section.

It is typically formulated as

$$
\begin{aligned}
[\mathbf{u}_t, \mathbf{r}_t]^\top &= \mathrm{sigm}(\mathbf{W}_z \mathbf{x}_t + \mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{b}_g) \\
\tilde{\mathbf{h}}_t &= \tanh(\mathbf{U}_z \mathbf{x}_t + \mathbf{U}_h (\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_h) \\
\mathbf{h}_t &= (\mathbf{1} - \mathbf{u}_t) \odot \mathbf{h}_{t-1} + \mathbf{u}_t \odot \tilde{\mathbf{h}}_t.
\end{aligned}
$$

Orthogonal weight matrices

Besides clipping, an alternative strategy that can be employed to reduce vanishing gradient is the initialization of the recurrent matrices to be orthogonal matrices. As orthogonal matrices have unit norm, multiplication by them when backpropagating gradients does not result in a lower gradient norm between one timestep compared to the previous one, disregarding effects of the activation function.

Orthogonal initializations were first proposed for feed-forward neural networks in Saxe et al. (2013), and later for RNNs in the context of GRUs in Cho et al. (2014). Later, Le et al. (2015) showed that orthogonal initializations make Elman-style RNNs with ReLU activation functions train with relative ease.

Some work goes beyond initializing weight matrices with orthogonal matrices to ensuring the weight matrices are *always* orthogonal, by choosing them to be a parameterized family and/or fixed set of orthogonal matrices (Arjovsky et al., 2016; Henaff et al., 2016; Jing et al., 2017, e.g.).

1.3.2 Spatiotemporal convolutions

Convolutional neural networks (CNNs), containing stacked convolutions operating spatially over an image, have been instrumental in advancing performance in computer visions tasks such as object recognition that receive an image as input (Krizhevsky et al., 2012). A basic 2D convolution layer from C channels to C' channels, without a bias and with unit stride for simplicity, computes

$$[\text{conv}(\mathbf{x},\mathbf{w})]_{c'ij} = \sum_{c=1}^{C}\sum_{i'=1}^{k_w}\sum_{j'=1}^{k_h} w_{c'ci'j'}x_{c,i+i',j+j'},$$

for input $\mathbf{x}$ and weights $\mathbf{w} \in \mathbb{R}^{C'\times C\times k_w\times k_h}$ where we define $x_{cij} = 0$ for i,j out of bounds. Spatiotemporal convolutional neural networks (STCNNs) can process video data by convolving across time, as well as the spatial dimensions (Karpathy et al., 2014; Ji et al., 2013). Hence similarly,

$$[\text{stconv}(\mathbf{x},\mathbf{w})]_{c'tij} = \sum_{c=1}^{C}\sum_{t'=1}^{k_t}\sum_{i'=1}^{k_w}\sum_{j'=1}^{k_h} w_{c'ct'i'j'}x_{c,t+t',i+i',j+j'}.$$

By convolving over both time and space at the same time, we produce a layer strictly powerful than convolving over just time and space alone—that is, if we set $w_{c'ct'i'j'} = 0$ (for all c', c, $t \neq \lfloor k_t/2 \rfloor$) we recover spatial convolutions, and if we set $w_{c'ctij} = 0$ (for all c', c, $i \neq \lfloor k_w/2 \rfloor$ and $j \neq \lfloor k_h/2 \rfloor$) we recover temporal convolutions. Instead of manually choosing spatial modelling (e.g. spatial convolutions) or temporal modelling (e.g. temporal 1-D convolutions or RNNs) at a specific point in a network, by using spatiotemporal convolutions we allow the data and learning process to dictate how much temporal modelling, spatial modelling, or a mix thereof occurs at each layer. Intuitively, spatiotemporal convolutions allow the model to compare pixels between frames across time (or feature maps for layers after the first), in addition to simultaneously learning spatial filters that compute spatial features.

All the other standard layers seen for 1-D and 2-D convolutions extend trivially to spatiotemporal convolutions, such as dropout, batchnorm and other kinds of normalization, and various types of pooling. However, due to the semantic difference between the temporal axis compared to spatial ones, the extents of each layer (e.g. kernel size or pooling window size) may be different in the time dimensions versus the spatial dimensions. Thus, when performing architecture search, these need not be tied together, as they tend to be for a spatial CNN.

1.3.3 Modeling sequences with neural networks

Sequences of discrete symbols can readily be modeled autoregressively in this framework, using feed-forward neural networks, RNNs, or even convolutional neural networks. In this section, for concreteness, we focus on RNNs.

Sequence-to-sequence modeling (seq2seq) refers to problem of mapping from a variable-length input sequence to a variable-length output sequence. Concretely, suppose we want to map from $\{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$ to $\{\mathbf{y}_1, \ldots, \mathbf{y}_U\}$, where T is the number of input timesteps which varies per input, and U is the number of output timesteps which varies as well.

Modeling this mapping probabilistically is typically formulated as

$$p(\mathbf{y}|\mathbf{x}) = \prod_{u=1}^{U} p(\mathbf{y}_u|\mathbf{y}_{1:u-1}, \mathbf{x}_{1:T}), \tag{1.3.3.1}$$

where the notation $x_{a:b}$ is sequence slice from a to b inclusive, i.e. $x_a, x_{a+1}, \ldots, x_{b-1}, x_b$.

Each term in the product is a distribution parameterized by the output of an RNN. For example, for modeling a sequence of real numbers one could use a Gaussian or Gaussian mixture model, or for producing a discrete sequence (such as words), one would use a categorical distribution. Each timestep of the RNN receives the previous timestep's output as an input, i.e. **autoregressively**. Such models are usually trained using maximum likelihood, with so-called "teacher forcing" where the ground-truth previous timestep is used (as opposed to one produced by the model).

RNN-based language modeling is also formulated as (1.3.3.1), but without the dependence on $\mathbf{x}$ (Mikolov et al., 2010b).

A common approach introduced by Sutskever et al. (2014) is to use an RNN to process the input and produce a fixed-sized vector, then use another RNN to produce an output sequence one timestep at a time. We refer to this as *encoder-decoder* seq2seq. Hence, in (1.3.3.1) each term on the left corresponds to one timestep of the output RNN. The input to the first timestep of the output RNN is the state or output from the last timestep of the input RNN.

This approach has two related shortcomings. First, the encoder produces a fixed-sized vector, regardless of the number of input timesteps. The encoding process is effectively compression, which may make representation of the input sequences difficult. Second, since there are long term dependencies that must propagate through the RNN, there are also long sequences through which gradients must propagate. This makes learning difficult as well.

These two shortcomings are both addressed with the *encoder-to-decoder with attention* model, introduced by Bahdanau et al. (2014). In this approach, instead of

compressing the input sequence by using only the last timestep of the encoder RNN as input to the decoder RNN, we keep each timestep's output from the encoder RNN, and each timestep of the decoder receives a weighted sum of the encoder's outputs. The weights of this weighted sum are called the attention weights. That is, the decoder produces a probability distribution over the timesteps it chooses to "attend" to. The timesteps with higher probability can be interpreted as the more important input timesteps for producing the current output timestep. Then, the weighted sum can be seen as an expectation of the encoder RNN's outputs with respect to this distribution. We do not go further into detail here as this approach is not used in this thesis, except as a baseline in Chapter 4. For further details on this attention-based approach, see the original paper or Goodfellow et al. (2016).

For the purpose of this thesis, we use "sequence-to-sequence" or "seq2seq" narrowly, referring specifically to this encoder-decoder approach. Similarly, "sequence-to-sequence with attention" refers to the attentional variant.

Neither seq2seq nor the attentional variant take advantage of a key property of the speech recognition problem: monotonicity. While the attentional variant may be modified to enforce monotonicity, later we will explore models with this property built-in, as this results in a simpler model. As we will see in some applications later, a key advantage of having separate AMs and LMs is ease of adapting to new domains by simply replacing the LM and lexicon, so we often explicitly do not want too much sequential modeling of the output labels in the acoustic model. This replacement is hard to perform in an end-to-end model or when the acoustic model models dependencies in the label sequences, as in seq2seq models.

In Section 1.5, we will look at connectionist temporal classification, a modeling approach for monotonic sequence transduction that does not require alignments.

1.4 Brief introduction to finite state transducers

An important tool in classical speech recognition and this thesis is the *finite state transducer* (FST), a type of finite state automata that both accepts input and produces output, useful for representing (weighted) mappings from strings to strings. In this section, we present a brief review of FSTs, assuming basic undergraduate-level knowledge of finite state automata, general machine learning, and basic natural language processing (e.g. n-gram language models). For a more extensive overview, see Mohri et al. (2002).

FSTs have long been an important tool for decoding in speech recognition (e.g. Mohri et al., 2002; Miao et al., 2015; McGraw et al., 2016). In decoding, we want to find the most likely sequence of words for a given utterance; this is a search problem. As briefly mentioned at the end of Section 1.2, the *decoding* step of ASR consists of mapping from phonemes to words using the *lexicon* and then weighing the word sequences by their probability a *language model*. Furthermore, search over the HMM states results in sequences of state IDs or GMM mixture component IDs, need to be converted to sequences of phonemes. We additionally may need to map from CD phonemes or triphones to plain phonemes. Conveniently, all of these pieces are readily expressed solely using FSTs, alleviating the need for model-specific procedures.

We begin this section with a review of finite state automata (FSAs), the weighted extension of FSAs (WFSAs), and finally weighted finite state transducers.

1.4.1 Finite state acceptors

First, we remind the reader that a *finite state acceptor* (FSA) is defined by a finite set of states Q, a finite symbol set Σ (also often called an *alphabet*), a set of transitions $T \subseteq Q \times \Sigma \times Q$ (i.e. consisting of a current state, symbol, and next state), an initial state $q_0 \in Q$, and a set of accepting states $A \subseteq Q$ (sometimes called *final states*).

An FSA can be interpreted as a directed graph, with vertices defined by Q and edges defined by T, and is frequently drawn as such. As an example, Figure 1.6 shows a simple FSA that generates the set of binary strings ($\Sigma = \{0,1\}$) that have an even number of zeros. Note that q_0 is an accepting state, as denoted by the double concentric circle, and is also the initial state as denoted by the dangling arrow to its left.

A *path* π of length n through an FSA is a sequence of transitions starting in state q_0 and ending in some state $q_n \in A$, where each subsequent state after the initial state follows a transition in T—that is, for all $i = 1, \ldots, n, (q_{i-1}, s_i, q_i) \in T$

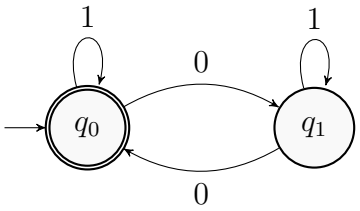


Figure 1.6: A simple FSA that accepts precisely the set of binary strings with an even number of zeros.

for some symbol s_i. A path is therefore associated with a string, consisting of symbols produced by each transition, $s_1, \ldots, s_n$. Hence, an FSA represents a set of strings (called a *language*) induced by the paths through its states and transitions. Note that multiple FSAs may accept exactly the same set of strings; this implies that FSAs can be transformed (e.g. to reduce the number of states) while preserving the language they accept.

Not all sets of strings can be represented by FSAs; those that can are called *regular languages*. This is due to the restriction that the number of states is finite; this rules out problems that involve counting, for instance, unless the counting is bounded (so not, for instance, the set of all natural numbers).

To check if an FSA accepts a given string, we traverse it. That is, we begin at the start state q_0, then follow the transitions in T that match the current state and the current symbol in the string, and repeat this latter step. If at any point there are no transitions matching the current state and symbol, the procedure is terminated early and the string is not accepted. After processing all the symbols in the string, the string is accepted if the state landed upon is an accepting state.

Non-deterministic FSAs

If there exist two transitions with the same first state and symbol, the FSA is called a *non-deterministic FSA*. In other words, at a given state, a non-deterministic FSA may have multiple next states that correspond to consuming the same symbol. We also introduce an ϵ symbol meaning 'no symbol' or 'empty string'; a transition with this symbol is followed without consuming any of the string. Thus, for non-deterministic FSAs, the transitions satisfy $T \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$.

For a non-deterministic FSA, to verify if the FSA accepts a given string, we must keep track of a *set* of active states, say S, rather than just a single current state. The set S begins as $\{q_0\}$. Then, if the next symbol to be consumed is c, the next set of active states is $\{q' : q \in S \text{ and } ((q, c, q') \in T \text{ or } (q,\epsilon,q') \in T)\}$. (The latter half of the 'or' doesn't consume c; thus, we also need to keep track of

how much of the string has been consumed at each active state.) That is, *each* current active state in S follows the transitions in T. After each symbol in the string has been followed, the string is accepted if and only if the active state set contains an accepting state, i.e. $S \cap A \neq \emptyset$.

It can be shown that a non-deterministic FSA can be transformed into a (larger) deterministic one, a state of which represents a *set* of active states in the non-deterministic FSA. This is called the powerset construction, see Sipser (2012, Theorem 1.39 on p. 55) for the full proof, which also allows ϵ symbols in the non-deterministic FSA. As the name suggests, in the worst case, the determinized FSA will have $2^{|Q|}$ states, one for each possible subset of Q. The operation of transforming a non-deterministic FSA into a deterministic one is called *determinization*. For an FSA F, its determinization is denoted $\det(F)$.

As any non-deterministic FSA can be determinized, non-deterministic FSAs are not more expressive than deterministic FSAs: they both represent the set of regular languages. For that reason, 'FSA' can hereafter refer to a non-deterministic FSA when unspecified.

Operations on FSAs

We just saw how a non-deterministic FSA can be determinized. Other operations on FSAs can be efficiently performed, such as union, intersection, concatenation, and repetition (also called closure, or Kleene star).

Each operation should be thought of as acting on the language that the FST represents. For instance, the union of two FSAs F_A and F_B, denoted $F_A \cup F_B$, results in an FSA accepting the union of the languages accepted by the two FSAs. The construction of $F_A \cup F_B$ is simple: create a non-deterministic FSA with an ϵ-transition to the start state of each FSA, as shown in Figure 1.7.

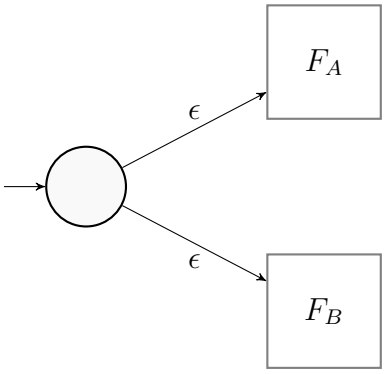


Figure 1.7: Constructing an FSA for the union $F_A \cup F_B$. The set of accepting states is the union of the sets of accepting states of F_A and F_B (not shown).

For the 'zero-or-more' repetition operation, consider an FSA F that accepts the language $\mathcal{A}$. The resulting FSA F^*, where * is called the Kleene star, accepts the following language:

$$\bigcup_{n=0}^{\infty} \text{concatenate}(a_1, \ldots, a_n), \text{ for } a_i \in \mathcal{A}$$

where 'concatenate' denotes string concatenation, with concatenate() defined to result in a set containing only the empty string. There are minor variations of this operator, such as the 'one-or-more' repetition of an FST F accepting language $\mathcal{A}$, denoted F^+. It accepts the language, $\bigcup_{n=1}^{\infty} \text{concatenate}(a_1, \ldots, a_n)$ (note the change to the lower limit of the union, $n = 1$ instead of $n = 0$).

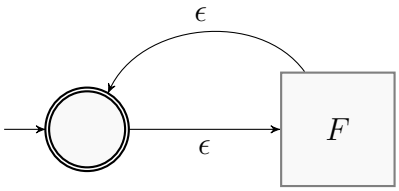


Figure 1.8: Constructing an FSA for the 'zero-or-more' repeat operation, F^* (Kleene star), of an FST F.

Another useful operation is *minimization*, the construction of an FSA representing the same language but with the minimum number of states. The algorithm is beyond the scope of this introduction, see Mohri et al. (2002). Minimization is commonly used as an optimization to reduce the size of an FST after performing operations on it. The minimization of an FST F is denoted $\min(F)$.

The above operations are all closed over the regular languages: their result is always a regular language. This should be clear, as the operands and results of each operation are all FSAs, and the existence of an FSA representing the language means the language is regular.

1.4.2 Weighted FSAs

A *weighted FSA* (WFSA) is an extension to the FSA definition above, where a weight is associated with each transition and accepting state. This allows strings to each be associated with a weight.

Formally, define a *semiring* to be a set along with:

- a commutative and associative addition operator $\oplus$,
- an associative multiplication operator $\otimes$,

- an associated identity for each operator ($\bar{0}$ and $\bar{1}$, respectively),
- distributivity of $\otimes$ over $\oplus$,[2]
- and where $\bar{0} \otimes x = x \otimes \bar{0} = \bar{0}$ for all x in the set.

Neither an additive inverse nor a multiplicative inverse is required.

Semiring	Set	$\oplus$	$\otimes$	$\bar{0}$	$\bar{1}$
Boolean	{0,1}	$\vee$	$\wedge$	0	1
Probability	$\mathbb{R}_+$	$+$	$\times$	0	1
Log (negative log prob.)	$\mathbb{R} \cup \{-\infty, +\infty\}$	$x,y \mapsto -\log(e^{-x} + e^{-y})$	$\times$	∞	0
Tropical	$\mathbb{R} \cup \{-\infty, +\infty\}$	min	$\times$	∞	0

Table 1.1: A list of some commonly-used semirings, see Allauzen et al. (2007) for more. Note that a WFSA with the boolean semiring reduces to a particular unweighted FSA.

A WFSA differs from a FSA by adding a semiring for weights, $K = (\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ ($\mathbb{K}$ is the set from which weights are drawn), redefining the transitions to include weights, $T \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times \mathbb{K} \times Q$, and associating a weight with each accepting state by defining a function $\rho : A \to \mathbb{K}$ (recall A is the set of accepting states). Here, we follow the notation of Mohri et al. (2002) but with minor changes, such as the lack of an initial weight–the resulting formulation is equivalent and is common in software implementations such as Allauzen et al. (2007).

The weight of a path π of length n is then the $\otimes$-product of the weights of the transitions on the path,

$$\text{weight}(\pi) = w_1 \otimes \ldots \otimes w_n \otimes \rho(q_n),$$

for each transition $t_i \in T$ with associated weight w_i and destination state q_i for $i = 1, \ldots, n$, where $q_n \in A$ (is an accepting state). The weight of a string s is then the $\oplus$-sum of the weights of all paths that produce the given string:

$$\text{weight}(\text{string } s) = \bigoplus_{\text{all paths } \pi \text{ producing } s} \text{weight}(\pi)$$

The operations mentioned for FSAs extend naturally to WFSAs. Another useful operation for our purposes is *shortest path*. The shortest path yields the string with the minimum weight, with respect to an order defined on the weight semiring. If the WFSA represents a probability distribution and the weights are interpreted as

[2]Depending on the operation, we may just need distributivity on the left, just on the right, or but for others we need both.

negative log probabilities, this results in the path associated with the string with the maximum probability. We will see an example of this later.

It is also possible to find the k-shortest paths efficiently, and also the n-lowest-cost strings (Mohri and Riley, 2002), often more useful in practice as many paths may correspond to the same string. The lowest cost *path* is the same for a particular FST with its weights treated as either in the tropical or log semirings, but the lowest cost *string* may not be, due to the use of the $\bigoplus$ operator.

1.4.3 Example: n-gram language models

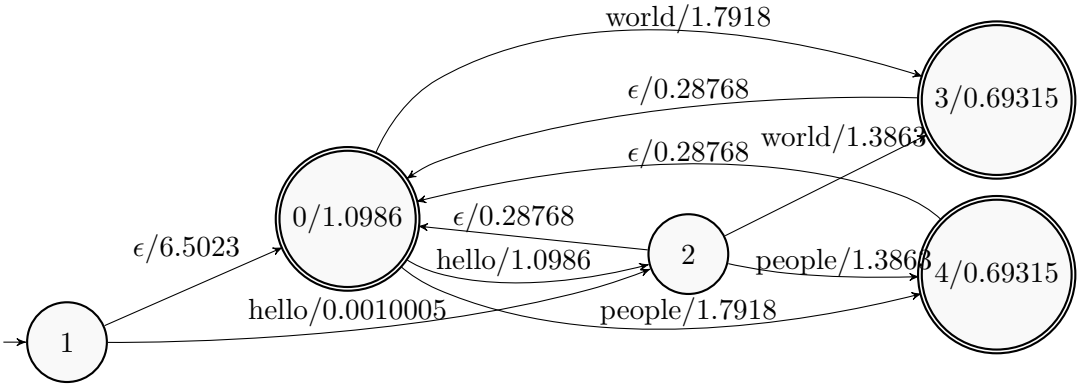
Probabilities (or negative log probabilities, simply a negative-log transformation of the former) form semirings allowing us to define WFSAs that represent probability distributions over strings. Using probabilities as weights (or equivalently, using negative log probabilities), we can represent an n-gram language model as an WFSA. If the weights are negative log probabilities, the numeric values of the weights can be kept and operators replaced with those of the tropical semiring, It should be intuitive that n-gram models fit into a finite-state framework: as n-gram models only look at a finite history, the history is encodable into finitely many states.

Specifically, one way to construct a WFSA for an n-gram language model is as follows. Consider a unigram w. There is a dedicated state for all unigrams from which transitions consume w, with the transition's weight set to the unigram probability, and where the transition leads to itself (since unigrams have no history). Now, consider the n-gram $w_1 \dots w_n$ ($n \geq 2$). Then, we add a transition from the state representing the history prefix $w_1 \dots w_{n-1}$, where the transition consumes w_n, and its weight is set to the n-gram's probability (scaled by the discounting factor, if applicable). The destination state is then set to the state representing the history $w_1 \dots w_n$ if such a prefix exists (i.e. there exists an n-gram $w_1 \dots w_n \bar{w}$ for some $\bar{w}$), or the state associated with $w_2 \dots w_n$ otherwise. Backoffs transitions go from state $w_1 \dots w_n$ to state $w_2 \dots w_n$, consuming nothing (i.e. ϵ), and with the backoff probability as the weight.

For an example, see Figure 1.9. For more details on this construction, see Roark et al. (2012, Figure 1), from OpenGRM, a library that takes this approach.

1.4.4 Weighted finite state transducers

As we saw in the previous section, weighted FSAs give us a framework through which we can express the cost or probabilities of strings, be they strings over characters, words, phonemes, or even non-language symbols. However, we would also like to



(a) Generated WFSA for the n-gram model. State 1 is the start state, and state 2, for instance, is the state that denotes the last word seen (a one-word history) was 'hello'. There are no 2-word histories here as the model does not have trigrams. Weights are all negative log-probabilities. Accepting state weights are denoted by a slash.

hello	0.3333333
hello world	0.25
hello people	0.25
world	0.1666667
world </s>	0.5
people	0.1666667
people </s>	0.5
</s>	0.3333333
<s>	0.3333333
<s>hello	0.999

(b) Uni-/bi-gram probabilities as generated by OpenGRM.

Figure 1.9: N-gram language model trained on two strings, "hello world" and "hello people", using OpenGRM (Roark et al., 2012) with an n-gram order of 2 (up to bigrams only) and Katz smoothing.

express *mappings* from strings to other strings. Weighted finite state transducers (WFSTs, hereon just FSTs) extend the idea of WFSA representing a relation, i.e. a collection of input-output pairs. Recall that unlike a function, a relation allows the same input to map to multiple (zero or more) outputs.

FSTs model this by labeling each transition with an input-output *pair*, as opposed to just a single symbol like (W)FSAs. To handle input lengths to differ from output lengths, the input and output are both allowed to be ϵ. Then, a path through this FST corresponds to a particular input string-output string pair—recall we use 'path' to mean a sequence of transitions starting at the initial state and ending in an accepting state. The weight of a path is defined in the same way as WFSAs: the $\otimes$-product of the weights at the transitions and the accepting state's weight.

The weight of an input-output string pair is defined to be the $\oplus$-sum of all paths with the given input-output string pair (ignoring ϵ symbols, of course).

The input and output strings need not come from the same alphabets: this allows us to, for instance, map from phoneme strings to word strings, as is done in ASR, or map from grapheme strings to other grapheme strings, as we will do for Hawaiian orthography in Chapter 2.

We can consider FSTs to be a generalization of WFSAs, which reduce to WFSAs when the input and output at each transition are the same. In practice, this is a common representation, allowing us to use a single software library for both.

Formal definition

More formally, we modify the earlier WFSA definition as follows. We now define both an input alphabet Σ and an output alphabet Γ, and transitions now contain both:

$$T \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \times \mathbb{K} \times Q.$$

To recap, that is the start state, input symbol, output symbol, weight, and target state. The weight of a path π is, like before, defined as

$$\text{weight}(\pi) = w_1 \otimes \ldots \otimes w_n \otimes \rho(q_n),$$

for each transition $(q_{i-1}, \sigma_i, \gamma_i, w_i, q_i) \in T$ for $i = 1, \ldots, n$, where $q_n \in A$ (is an accepting state). The weight of an input-output string pair $(a, b) \in \Sigma^* \times \Gamma^*$ is then the $\oplus$-sum of the weights of all paths that produce the given string:

$$\text{weight}(a,b) = \bigoplus_{\substack{\text{paths } \pi: \\ \text{remove-}\epsilon(\sigma_1 \ldots \sigma_n)=a, \\ \text{remove-}\epsilon(\gamma_1 \ldots \gamma_n)=b}} \text{weight}(\pi),$$

where remove-$\epsilon(s)$ removes each ϵ from the string s.

Operations on FSTs

The minimization and determinization operations for WFSAs are applicable to FSTs as well. Determinization, however, will not terminate on certain inputs. There are large classes of inputs for which it *does* terminate (Mohri, 1997), and workarounds for cases where it does not. This discussion is beyond the scope of this introduction, see Mohri and Riley (2002) for details.

Another simple but useful operation is *projection*. Input (output) projection deletes the output (input) symbols on an FST, resulting in a WFSA.

The most interesting operation that can be done with FSTs is *composition*, which works similarly to composition of functions (remember, FSTs represent relations,

not functions). Consider the composition to two FSTs F_A and F_B, denoted $F_A \circ F_B$. Of course, the output alphabet of F_A must be the same as the input alphabet of F_B, and their weights should be defined over the same semiring. Imagine an input string fed to F_A; it will produce a set of output strings. This set of output strings can then be fed as input to F_B, resulting in a set of its output strings.

Formally, if F_A transduces an input-output string pair (x, y) with weight weight(x,y), and F_B transduces an input-output string pair (y, z) with weight weight(y,z), their composition $F_A \circ F_B$ transduces (x, z) with weight weight(x,y) $\otimes$ weight(y,z). See Mohri et al. (2008) for the algorithm; faster algorithms exist for special cases (Allauzen et al., 2010). In the worst case, , the runtime of the standard algorithm is quadratic in transition counts of F_A and F_B: $O(|T_A||T_B|)$.

If FSTs F_A and F_B both happen to be WFSAs, their composition ends up being a special case of an operation on WFSAs called intersection. More explicitly, F_A accepts the string x with weight w, and F_B accepts the string x with weight w', $F_A \cap F_B$ accepts the string x with weight $w \otimes w'$.

In practice, many FST operations have 'on-the-fly' variants. This may be useful when using an FST in a search procedure or when computing its shortest path, in which case most states would never be visited. This reduces the need for precompiling a potentially large FST.

1.4.5 Example: speech recognition

FSTs are used in speech recognition for decoding.

First, we use an FST to represent probability distributions by ensuring $\sum_b$ weight(a,b), i.e. that all outputs for a given input $\oplus$-sum to $\bar{1}$ (in the probability or log semirings). For convenient handling of such an FST, we may want the property that the sum of outgoing transitions from a state have weights that add to $\bar{1}$ (in the probability or log semiring), at least approximately. While we perform relevant FST operations in the log semiring, we still continue to use the tropical semiring for search, as we will see shortly.

A decoder FST F, often called a *decoder graph*, will often be of the form:

$$F = a \circ (H \circ C \circ L^* \circ G), \tag{1.4.5.1}$$

where:

- a denotes the input acceptor, and generates HMM state IDs or GMM mixture component IDs,

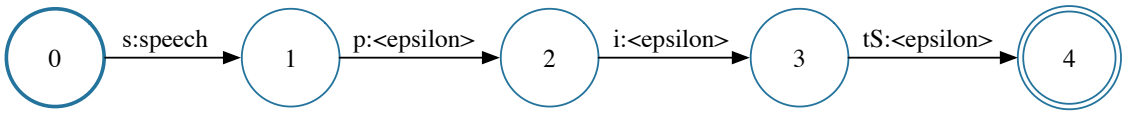


Figure 1.10: FST mapping from the pronunciation of "speech" to the word "speech".

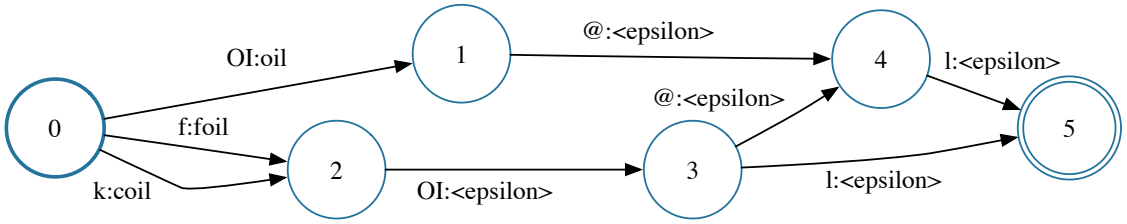


Figure 1.11: Lexicon FST mapping from the pronunciation of several words to the words themselves: "foil", "coil", and "oil".

- H denotes the 'HMM' transducer, and maps from the above IDs to the corresponding context-dependent phoneme symbol,
- C denotes the 'context' transducer, and maps from context-dependent phonemes to the underlying 'raw' phoneme,
- L denotes the lexicon transducer, and maps from a pronunciation of a word to the word (L^* hence maps from a string of phonemes to string(s) of words rather than just a single word),
- and G denotes the 'grammar' acceptor, typically an n-gram language model over words.

The output symbol set of each component is compatible with the input symbol set of the next in the composition chain.

The construction of H and C is simple so we will not elaborate further, and G is an n-gram WFSA and has been discussed earlier. Instead, we focus on the lexicon. The lexicon's input symbol set is the set of phonemes, and its output symbol set is the set of words. It can be formed by taking individual word pronunciations as chain-shaped FSTs that map one pronunciation (string of phonemes) to one word, such as in Figure 1.10, and then taking the union of all of these chains. After computing the determinization and minimization, we get a rather small graph like Figure 1.11.

A few implementation details are omitted in (1.4.5.1) for simplicity. Composition of the parenthesized portion is performed from right to left—we disregard a for now. Also, composition is usually followed by determinization and then minimization to produce a small, deterministic FST. This means, as briefly mentioned above, that we need to ensure $L^* \circ G$ can be determinized. In general, it cannot be

determinized, but a simple preprocessing step called 'disambiguation' can be applied to L beforehand to make the composition determinizable, this procedure is beyond the scope of this discussion.

The first composition in (1.4.5.1), i.e. of a with the parenthesized part, is typically computed on-the-fly. The FST F gives a relation mapping from HMM state ID strings to word strings. We can optionally first compute the output projection, so we have a WFSA over words. The shortest path through F gives the highest probability word string.

1.4.6 Example: modernizing Hawaiian orthography

This section is a brief preview of Chapter 2.

Old Hawaiian missionary-era orthography was missing characters for long vowels and glottal stops, using the same characters for the short and long vowels, and completely omitting the latter. For instance, *ʻāina* means land and *ʻaina* means food, but both are transcribed as 'aina' in the old orthography. Word boundaries have occasionally changed over time as well.

We can model this as a transducer F:

$$F = I_s \circ C \circ G,$$

where:

- I_s is the old orthography input acceptor for old Hawaiian input string s (simply a chain-shaped acceptor that accepts a fixed input string in the old Hawaiian orthography),
- C is a orthography-change transducer that we use to incorporate our knowledge of how Hawaiian might have changed,
- and G is a character-level language model.

C serves the purpose of generating a space of candidates that G then narrows. For instance, it may (optionally) map 'a' to 'ā' and optionally insert glottal stops in appropriate locations. The version in Chapter 2 is more elaborate, also allowing word boundaries to change. As there are these 'optional' transitions (i.e. 'a' to 'a' and 'a' to 'ā'), C is non-deterministic; F is composed from left to right. This is cheap as I_s is small. One could also insert a word dictionary after C and make G a word-level LM, but in this case, there is insufficient data for such a word-level approach. The shortest path through F then results in the most probable new orthography prediction for input I, according to the language model. No pieces here are trained except G, so we do not require any paired (old, new) Hawaiian training data.

1.5 Connectionist temporal classification

The connectionist temporal classification (CTC) structured loss function (Graves et al., 2006) is widely used in modern speech recognition as it eliminates the need for alignments that say where in the input each part of the label sequence occurs (Amodei et al., 2016; Graves and Jaitly, 2014; Maas et al., 2015). Given a model that outputs a sequence of discrete distributions over a label set augmented with a special "blank" token, CTC computes the probability of a sequence by marginalizing over all sequences that are defined as equivalent to this sequence. As an example, for a model with 3 output timesteps, the CTC loss would consider "aam", "amm", "␣am", "a␣m", and "␣am" to all be equivalent, where '␣' represents the CTC blank symbol.

Notice from this example that CTC both introduces a blank symbol, giving the model the option of not outputting a symbol at a particular position, and collapses duplicates, allowing the neural network to output the same symbol multiple times in a row—in some applications, a label may occur over a span of timesteps rather than instantaneously, such as in optical character recognition and speech recognition. The marginalization of the alignments in the loss function allows CTC to simultaneously remove the need for alignments and address the problem of variable-length sequences.

CTC is an example of a sequential neural network model with latent variables, where efficient marginalization of the latent variables is possible via dynamic programming (Yu et al., 2016; Kim et al., 2017; Eisner, 2016). In the case of CTC, these latent variables represent a "true" alignment of where labels occur in an input sequence. Since we don't know what the true alignment is, we sum over *all* possible alignments.

Let V denote the labels set that the model classifies at a single timestep of its output, then $\bar{V} = V \cup \{␣\}$ defines a blank-augmented vocabulary, where ␣ denotes the CTC blank symbol. Given an input $\mathbf{x}$ (typically a sequence), a neural network with parameters $\boldsymbol{\theta}$ defines a distribution $p(z_t|\mathbf{x}_{1:T};\boldsymbol{\theta})$ over the discrete labels $z_t \in \bar{V}$. Define the *collapsing* function $\mathcal{B}_T : \bar{V}^T \to V^*$ that, given a length-T string over $\bar{V}$, deletes duplicate adjacent symbols, and then removes blank symbols, producing a string over V of length $U \leq T$. Then, CTC defines the probability of a length-U label sequence $\mathbf{y}_{1:U}$ by marginalizing over all possible alignment sequences $\mathbf{z}_{1:T} \in \bar{V}^T$:

$$p(\mathbf{y}_{1:U}|\mathbf{x}_{1:T}) = \sum_{\mathbf{z}_{1:T} \in \bar{V}^T} p(\mathbf{y}_{1:U}|\mathbf{z}_{1:T}) p(\mathbf{z}_{1:T}|\mathbf{x}_{1:T};\boldsymbol{\theta}) \tag{1.5.0.1}$$

$$= \sum_{\mathbf{z}_{1:T} \in \bar{V}^T} p(\mathbf{y}_{1:U}|\mathbf{z}_{1:T}) \prod_{t=1}^{T} p(z_t|\mathbf{x}_{1:T};\boldsymbol{\theta}) \tag{1.5.0.2}$$

where we define

$$p(\mathbf{y}_{1:U}|\mathbf{z}_{1:T}) = \begin{cases} 1 & \text{if } \mathcal{B}_T(\mathbf{z}_{1:T}) = \mathbf{y}_{1:U}, \\ 0 & \text{otherwise.} \end{cases} \tag{1.5.0.3}$$

This can be seen to be equivalent to the original presentation of Graves et al. (2006) and Graves (2012b), by combining Equations (1.5.0.1) and (1.5.0.3) to give

$$p(\mathbf{y}_{1:U}|\mathbf{x}_{1:T}) = \sum_{\mathbf{z}_{1:T} \in \mathcal{B}_T^{-1}(\mathbf{y}_{1:U})} \prod_{t=1}^{T} p(z_t|\mathbf{x}_{1:T}; \boldsymbol{\theta}). \tag{1.5.0.4}$$

In this form, however, it is not as obviously a latent variable model where the latent variable represents the alignment. For example, if $T = 3$, CTC defines the probability of a string "am" as the sum of all possible alignments to this string: $p(aam) + p(amm) + p(_am) + p(a_m) + p(am_)$. Note that there exist many $\mathbf{y}$ that do not have a valid alignment $\mathbf{z}_{1:T}$; the simplest example is when $\mathbf{y}$ is more than T steps long.

CTC has conditionally independent outputs, as seen in Equation (1.5.0.2). This may be considered a positive feature when using CTC to represent an acoustic model, as we want the decoder graph and language model to handle disambiguation and long-term sequence modeling, rather than let this functionality creep into the acoustic model.

1.5.1 Probability of a particular sequence

To perform maximum likelihood, we need to be able to compute the probability of a particular label sequence, as in Equation (1.5.0.2). For notational conciseness, we drop the dependence on $\mathbf{x}$ and $\boldsymbol{\theta}$. There are $|\mathcal{B}_T^{-1}(\mathbf{y})| = 2^{T-U^2+U(T-3)}3^{(U-1)(T-U)-2}$ values of $\mathbf{z}_{1:T}$ with $p(\mathbf{y}_{1:U}|\mathbf{z}_{1:T}) = 1$, which is exponential in T (Graves, 2012b, §7.3). Naively enumerating all terms of the sum is intractable. Thus, a dynamic programming algorithm was proposed in Graves et al. (2006). We change the notation from that of the original work for clarity.

First, suppose we have a label sequence $\mathbf{y}$ with length U. Then, we can construct a recurrence on the probability of all of the length-t (for $t \leq T$) alignment prefixes that map to $\mathbf{y}_{1:u}$, which we call the *forward probabilities*.

Let $\alpha_{t,u,b}$, where $1 \leq t \leq T$, $1 \leq u \leq U$, and $b \in \{0 \text{ (false)}, 1 \text{ (true)}\}$ (think of b as a boolean for 'blank'), denote the probability that the alignment $\mathbf{z}_{1:t}$ produces

the label prefix $\mathbf{y}_{1:u}$ where the last symbol of the alignment is either non-blank (i.e. y_u, when $b = 0$), or blank (when $b = 1$). That is, we define

$$\alpha_{t,u,0} := P(\mathcal{B}_t(\mathbf{z}_{1:t}) = \mathbf{y}_{1:u} \text{ and } z_t = y_u), \quad \text{pr. ending in non-blank } (b = 0) \tag{1.5.1.1}$$

$$\alpha_{t,u,1} := P(\mathcal{B}_t(\mathbf{z}_{1:t}) = \mathbf{y}_{1:u} \text{ and } z_t = \sqcup). \quad \text{pr. ending in blank } (b = 1) \tag{1.5.1.2}$$

As a reminder, $\mathbf{z}$ is the random variable, and $\mathbf{y}$ is a fixed constant string. We have several simple base cases:

$$\alpha_{1,0,1} = p(z_1 = \sqcup),$$
length-1 alignment has empty string ending in blank
$$\alpha_{1,1,0} = p(z_1 = y_1),$$
length-1 alignment has first symbol of $\mathbf{y}$
$$\alpha_{1,u,b} = 0. \qquad (\text{for } u \geq 2 \text{ and } b \in \{0,1\})$$
length-1 alignment has more than 1 symbol of $\mathbf{y}$

For any $t > 1$ and $u \geq 1$, we can partition $\mathcal{B}_t^{-1}(\mathbf{y}_{1:u})$ such that we write it recursively in terms of smaller t and smaller-or-equal u, in turn giving a recurrence for $\alpha_{t,u,b}$. Thus, we have, for the alignment ending in a non-blank ($b = 0$):

$$\alpha_{t,u,0} = p(z_t = y_u) \times \begin{cases} \overbrace{\alpha_{t-1,\,u-1,\,1}}^{\substack{\text{need blank} \\ \text{due to duplicate}}} + \overbrace{\alpha_{t-1,\,u,\,0}}^{\substack{\text{or instead, don't} \\ \text{consume any of } \mathbf{y}}} & \text{if } y_u = y_{u-1} \text{ (duplicate)} \\ \underbrace{\alpha_{t-1,\,u-1,\,0}}_{\text{non-blank}} + \underbrace{\alpha_{t-1,\,u-1,\,1}}_{\text{blank}} + \underbrace{\alpha_{t-1,\,u,\,0}}_{\text{don't consume any of } \mathbf{y}} & \text{otherwise} \end{cases}$$

For $b = 1$, where the alignment ends in a blank, we have:

$$\alpha_{t,u,1} = p(z_t = y_u)\big(\underbrace{\alpha_{t-1,\,u,\,0}}_{\substack{\text{alignment ends} \\ \text{in non-blank}}} + \underbrace{\alpha_{t-1,\,u,\,1}}_{\substack{\text{alignment ends} \\ \text{in blank}}} \big),$$

noting that, of course, we do not consume any of $\mathbf{y}$ in these cases. The probability of the full label sequence $\mathbf{y}$ (with length U) is then given by

$$\begin{aligned} p(\mathbf{y}|\mathbf{x};\theta) &= \sum_{\mathbf{z}_{1:T} \in \mathcal{B}_T^{-1}(\mathbf{y}_{1:U})} \prod_{t=1}^{T} p(z_t|\mathbf{x}_{1:T};\boldsymbol{\theta}) \qquad \text{by (1.5.0.4)} \\ &= P(\mathcal{B}_T(\mathbf{z}_{1:T}) = \mathbf{y}_{1:U}) \\ &= P(\mathcal{B}_T(\mathbf{z}_{1:T}) = \mathbf{y}_{1:U} \text{ and } z_T = \sqcup) + P(\mathcal{B}_T(\mathbf{z}_{1:T}) = \mathbf{y}_{1:U} \text{ and } z_T \neq \sqcup) \\ &= \alpha_{T,U,1} + \alpha_{T,U,0}, \qquad \text{by (1.5.1.1) and (1.5.1.2)} \end{aligned}$$

since the alignment can end in either a blank ($b = 1$) or non-blank ($b = 0$). Therefore, this yields a dynamic programming algorithm for computing the probability of a particular label sequence $\mathbf{y}$, i.e. $p(\mathbf{y}|\mathbf{x};\theta)$, and hence the negative log likelihood needed for training. Actual implementations evaluate all of the above recurrences in log space (using log probabilities) to prevent underflow.

The above dynamic programming procedure can be differentiated with respect to $p(z_i|\mathbf{x};\boldsymbol{\theta})$ and hence $\boldsymbol{\theta}$; reverse-mode automatic differentiation guarantees there exists a procedure for computing the gradient that is at most five times slower than the forward pass (Griewank and Walther, 2008). In other words, since the above algorithm for computing $p(\mathbf{y}|\mathbf{x};\theta)$ runs in $O(T{\cdot}U)$ time, the automatic differentiation-based computation of $\nabla_\theta p(\mathbf{y}|\mathbf{x};\theta)$ also runs in $O(T \cdot U)$ time. Alternatively, Graves et al. (2006) and Graves (2012b) suggest an alternative procedure also recursively computes a *backward probability* defined similarly to α, and expresses the derivative with respect to the logits in terms of both the forwards and backwards probabilities.

1.5.2 Sampling CTC outputs

Let $q(\mathbf{y})$ be a CTC distribution. Our goal is to sample $\mathbf{y} \sim q$.

Motivation. Suppose we want to compute

$$\mathbb{E}_{y\sim q(\cdot|x)}[-\log p(\mathbf{y}|\mathbf{x})]. \tag{1.5.2.1}$$

Such a computation may arise, for instance, if we want to distill one CTC model into another model (Hinton et al., 2015). If q denotes an audio speech recognition model used as a 'teacher', and p denotes a visual speech recognition 'student' model, Equation (1.5.2.1) allows us to perform cross-modality distillation, the transfer of the predictive distribution from the teacher to the student. This is extremely useful: if we have a large amount of audio-video pairs (without transcripts), we can transfer the predictive distribution from the audio speech recognition model (the audio modality) to the visual speech recognition (the visual modality). Although we might consider simply getting the top prediction from the audio CTC distribution and using this as a target label sequence for the visual model, if the audio model is insufficiently confident, its best mode may not be very informative. The full distribution contains much more information; Hinton et al. (2015) refers to making use of this as training with "soft targets". Equation (1.5.2.1) cannot be computed in closed form, but it can be approximated if we can sample from $q(\cdot|x)$. The example described here is related to cross-modal supervision transfer in Gupta et al. (2015), but their work does not handle sequences; Kim and Rush (2016) performs distillation on (non-CTC) sequence models, but for machine translation, and not across modalities.

Another application is training a model using the expectation of a non-differentiable loss with respect to the model's predictive distribution as an objective. For instance, Graves and Jaitly (2014) trains using the following objective, where p denotes the CTC model being trained:

$$\mathbb{E}_{\mathbf{y}\sim p(\cdot|\mathbf{x};\boldsymbol{\theta})}[\text{word-error-rate}(\mathbf{y}, \mathbf{y}_{\text{target}})],$$

where word-error-rate is a nondifferentiable scalar evaluation metric. The gradient of this expectation with respect to $\boldsymbol{\theta}$ can be computed using the score function estimator (Fu, 2006), which requires sampling from the CTC distribution.

Sampling. An ancestral sampling procedure for sampling $\mathbf{y}$ can be constructed as follows: for $t = 1, \ldots, T$, sample the categorical variables z_t by Equation (1.5.0.2); then set $\mathbf{y} = \mathcal{B}_T(\mathbf{z}_{1:T})$ with probability 1 by Equation (1.5.0.3). Since CTC defines z_t as conditionally independent, the timesteps can be sampled in parallel, without requiring calling any neural network operations within the sampling procedure. This CTC sampling procedure is described in Graves and Jaitly (2014).

Sampling from RNN decoders as used in sequence-to-sequence, RNN transducers, and other similar RNN models all require expensive neural network operations in between ancestral sampling steps on the output tokens; that is, both sampling and marginalization of the alignments require evaluating an RNN as a subroutine (Sutskever et al., 2014; Yu et al., 2016; Graves, 2012a). In other words, it is expensive to obtain a large number of samples for these models, whereas it is very cheap for CTC.

1.5.3 Prediction

At test time, for a CTC distribution $p(\mathbf{y}|\mathbf{x})$, we want to compute the most likely prediction:

$$\mathbf{y}^* = \arg\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}).$$

It is not obvious how to efficiently and exactly find $\mathbf{y}^*$. A naive solution, to enumerate all alignments and marginalize, is prohibitively expensive.

Top-1 alignment. A quick approximation computes the most probable alignment, then feeds it through $\mathcal{B}_T$ to turn it into a label sequence:

$$\begin{aligned}\mathbf{y}^* &\approx \mathcal{B}_T(\arg\max_{\mathbf{z}_{1:T}} p(\mathbf{z}_{1:T}|\mathbf{x},\boldsymbol{\theta})) \\ &= \mathcal{B}_T(\arg\max_{z_1} p(z_1|\mathbf{x},\boldsymbol{\theta}) \cdots \arg\max_{z_T} p(z_T|\mathbf{x},\boldsymbol{\theta})).\end{aligned}$$

This approximation is accurate when one alignment receives most of the probability mass, which is sometimes true but not always.

Beam search. A more accurate approximation can be computed by constructing a beam search procedure for CTC, as described in Graves and Jaitly (2014). Beam search is a breadth-first search procedure where only the top few search candidates are kept at each iteration (either as a fixed number of candidates, or within a fixed margin of the best cost, or both). As with the dynamic programming algorithm for the computation of αs described earlier, it is important to separately keep track of whether the previous symbol is a blank or non-blank. Doing this requires careful bookkeeping but is otherwise a standard beam search algorithm.

Specifically, we proceed as follows. Each element of the beam is a triple: $(\text{labels}, p_{\text{blank}}, p_{\text{non-blank}})$. Note that 'labels' is defined over V, not $\bar{V}$, and p_{blank} denotes the probability that alignment represents 'labels' and ending in a blank, and $p_{\text{non-blank}}$ is defined similarly.

We repeatedly expand each element of the beam by one new timestep of z, correspondingly updating the label sequence (unless expanding by the blank or a duplicate), and the two probabilities (being careful to handle blanks and duplicates similarly to the recursion for α). If two beam elements result in the same string, we merge them, adding their probabilities. After expanding all the beam elements by one timestep of z, we sort the beam by its total probability (i.e. $p_{\text{blank}}+p_{\text{non-blank}}$) and discard beam elements with low probability. These steps are described in Figure 1.12.

The result is the beam element with the highest value of $p_{\text{blank}} + p_{\text{non-blank}}$. Because of the merge step denoted (*), we can interpret this as approximately performing marginalization. As the beam width increases, the approximation improves; this lets us trade off accuracy and computation.

1.5.4 Decoding with CTC and FSTs

Suppose we have a CTC model that produces phonemes, and a FST decoder graph D that maps from phoneme sequences to word sequences, encapsulating the lexicon and language model. There are two ways we can approach decoding words using these two models. The first is to express the CTC model's predictive distribution as a WFSA, and the second is to perform CTC beam search while simultaneously feeding the resulting phoneme sequences through the FST.

FST-only approach. For the first approach, we first construct a WFSA that represents the CTC model's predictive distribution, called C. Each state in this acceptor corresponds to one timestep of the model, for a total of T states. Each

Figure 1.12: Pseudocode for CTC beam search.

```
beam ← (‘’, p_blank = 1, p_non-blank = 0)    ▷ single initial beam element
for each each CTC timestep (timesteps of z) do
  new beam ← copy of beam
  for each beam element (s, p_blank, p_non-blank) in beam, symbol c do
    ▷ Expand beam element by one new timestep of z using symbol c:
    Compute p'_blank, p'_non-blank using p_blank, p_non-blank, c, and z
                ▷ similar to α recursion
    if c is blank or duplicate then
      s' ← s
    else
      s' ← concatenation of s and c
    if s' ∈ new beam then
      Combine beam elements by adding probabilities (*)
    else
      Add to new beam
  Sort new beam by p_blank + p_non-blank
  beam ← top K elements of new beam
```

transition from the tth state to the $(t+1)$th state, emitting symbol $a \in \bar{V}$, is assigned a weight of $p(z_t = a)$.

Next, we construct a CTC preprocessing FST P: this (unweighted) FST performs the operation specified by $\mathcal{B}_T$, mapping from $\bar{V}$ to V.

Finally, shortest-path$(C \circ P \circ D)$, where D denotes the full decoder graph, produces the top word prediction. Note that if the tropical semiring is used, the most probable alignment from the CTC distribution is used rather than performing proper marginalization.

CTC-FST beam search approach. This approach is a modification of standard beam search. Suppose the decoder graph FST D is a non-deterministic FST with state set Q, input alphabet V (typically phonemes), and output alphabet W (typically words), with weights over $\mathbb{R} \cup \{\infty\}$ (tropical semiring). Each element of the beam is now defined as $(\text{labels}, p_{\text{blank}}, p_{\text{non-blank}}, \text{fst-states})$.

The new ‘fst-states’ value holds a set of tuples, $(q, \text{words}, i, \text{path-weight}) \in Q \times W^* \times \{\text{true, false}\} \times (\mathbb{R} \cup \{\infty\})$, each keeping track of an individual active state in the FST. Recall that when traversing non-deterministic automata, multiple states may be active at the same time. The i variable is a boolean denoting whether the FST has consumed the label symbol yet.

The algorithm itself proceeds similarly to normal CTC beam search, only now, each time we encounter a new label symbol, we also feed it through D to obtain a new set of active states. This function takes as input ‘fst-states’ and the new

label symbol (call it y), and returns a new set of 'fst-states'. First, for each element of 'fst-states', we initialize i to false. Then, using any graph traversal algorithm such as depth-first search, we follow the transitions of the FST that either input ϵ (regardless of whether i is true) or the label y (only when i is false), simultaneously keeping track of the state, words, and cumulative path weight. The boolean i ensures we only process the input symbol once, but still allow an arbitrary number of input epsilons. If too many input epsilons are encountered in a row, that partial path is pruned as its weight would just increase indefinitely, and so cannot be an optimal path. This ensures that the traversal does not infinitely loop and produce infinitely many sub-optimal paths.

When pruning the beam, we use the sum of the CTC negative log probability and the best path weight (i.e. the language model's negative log probability), and keep the lowest (best) K. We may optionally weigh the FST term to increase or decrease the influence of the language model.

We develop an algorithm loosely based on this one in Chapter 5, but where the outer loop loops over words instead of CTC timesteps. That guarantees that all beam elements contain the same number of words.

1.5.5 Conclusions

This section overviewed connectionist temporal classification and several associated algorithms. The following chapters will apply the techniques covered here.

2

Restoring Hawaiian orthography using FSTs and a hybrid FST/RNN approach

Abstract

In contrast to the older writing system of the 19th century, modern Hawaiian orthography employs characters for long vowels and glottal stops. These extra characters account for about one-third of the phonemes in Hawaiian, so including them makes a big difference to reading comprehension and pronunciation. However, transliterating between older and newer texts is a laborious task when performed manually. We introduce two related methods to help solve this transliteration problem automatically, given that there were not enough data to train an end-to-end deep learning model. One method is implemented, end-to-end, using finite state transducers (FSTs). The other is a hybrid deep learning approach which approximately computes the minimum cost path of the composition of an FST and an recurrent neural network (RNN) language model (equivalently, the intersection of a weighted finite state acceptor and an RNN language model). We find that the hybrid approach outperforms the end-to-end FST by partitioning the original problem into one part that can be modelled by hand, using an FST, and into another part, which is easily solved by an RNN trained on the available data.

2.1 Introduction

From 1834 to 1948, more than 125,000 newspaper pages were published in the Hawaiian language (Nogelmeier, 2010). Yet by 1981, many expected this once flourishing language to die (Benton, 1981). Hawaiian has since defied expectations

and experienced the beginnings of a remarkable recovery (Warner, 2001; Wilson and Kamanā, 2001). However much of the literary inheritance that is contained in the newspapers has become difficult for modern Hawaiians to read, since the newspapers were written in an orthography that failed to represent about one-third of the language's phonemes. This orthography, which we will refer to as the *missionary orthography*, excluded Hawaiian phonemes that did not have equivalents in American English (see Schütz, 1994), namely long vowels /iː eː aː oː uː/ and glottal stop /ʔ/. By contrast, the *modern Hawaiian orthography*, an innovation of Pukui and Elbert's Hawaiian dictionary (Pukui and Elbert, 1957), presents a nearly perfect, one-to-one mapping between graphemes and phonemes. The process of manual transliteration from missionary to modern Hawaiian orthography is extremely labor intensive. Yet the cultural benefits are so great that hundreds of pages of newspaper-serials have already been transliterated by hand, such as Nogelmeier's new edition of the epic tale of *Hiʻiakaikapoliopele*, the volcano goddess's sister (Hoʻoulumāhiehie, 2007). Critically important as such efforts are to the continued revitalization of this endangered language, they are still only a small sample of the material that could be made available to a modern Hawaiian audience.

In this chapter, we propose to automate, or semi-automate, the transliteration of old Hawaiian texts into the modern orthography. Following a brief review of related work (Section 2.2), we begin by describing a dataset of modern Hawaiian (Section 2.3). In Section 2.4, we present two methods for recovering missing graphemes (and hence phonemes) from the missionary orthography. The first composes a series of weighted FSTs; the second approximately "composes" an FST with a recurrent neural network language model (RNNLM) using a beam search procedure. Both approaches use only modern Hawaiian texts for training, which are much more plentiful than parallel corpora. Section 2.5 reports the results of our transliteration experiments using a simulated parallel corpus, as well as two 19th century newspaper articles for which we also have modern Hawaiian transcriptions. Being based on FSTs, both approaches are modular and extensible. We observe useful and promising results for both of our methods, with the best results obtained by the hybrid FST-RNNLM. These results showcase the strength of combining established hand-engineering methods with deep learning in a smaller data regime, with practical applications for an endangered language.

Notably, the approach described in this work is quite similar to the way automatic speech recognition has factored the audio feature to word sequence mapping problem into one that is modelled first using a learned acoustic model, followed by a much more handcrafted decoder typically including a pronunciation model and a

learned language model (Mohri et al., 2002). This allows hand-designed expert knowledge, words and their pronunciations, to be encoded into the model instead of learned from scratch inside of an end-to-end model that is trained to predict graphemes. The language model, which comes after the pronunciation model, provides disambiguation for homophones (words with the same pronunciation), and in the presence of phoneme recognition uncertainty. In this work, we employ ideas similar to these from speech recognition, but with the slightly different goal of constructing a model with no paired data. As in speech recognition, the language model provides disambiguation, but this time over a large space of possible orthography changes.

2.2 Related work

Many of the themes that we address relate to existing literature. For example, Hajič et al. (2000) and Scannell (2014) have written on machine translation (MT) for closely related languages and on multilingual text normalization, respectively. Though language-relatedness makes MT easier (Kolovratník et al., 2010), state-of-the-art techniques such as neural machine translation (NMT) have not performed well for languages with little data (Östling and Tiedemann, 2017). So while the Hawaiian transliteration problem could be cast as an instance of MT or of NMT, we chose to sidestep the scarcity of parallel data by not considering such approaches.

Hybrid approaches, which combine expert knowledge for well-understood structures with deep learning for data-plentiful subproblems, offer rich opportunities for data-efficient modelling. In particular, prior work has explored ways to combine FSTs and RNNs. For example, Sproat and Jaitly (2016) used an FST to restrict the search space when decoding from an RNN; Rastogi et al. (2016) incorporated RNN information into an FST. However, these approaches differ from the approximate FST-to-RNN composition algorithm that we introduce here (in Section 2.4.2).

2.3 Data

2.3.1 Phonemes & the modern orthography

Ignoring case, there is a neat mapping between the modern Hawaiian orthography and the Hawaiian phonemic inventory. The phonemic inventory contains eight consonants /h k l m n p v ʔ/ and ten vowels, of which five are short /a e i o u/ and five are long /aː eː iː oː uː/ (Parker Jones, 2018).

The consonants map onto the orthographic symbols ⟨H h K k L l M m N n P p W w ʻ⟩, where we give the upper- and lower-case variants in adjacent pairs: ⟨H h⟩ for /h/, ⟨K k⟩ for /k/, …, ⟨W w⟩ for /v/. An exception, the symbol ⟨ʻ⟩ has only one variant which maps to /ʔ/. The vowels map onto the symbols: ⟨A a E e I i O o U u Ā ā Ē ē Ī ī Ō ō Ū ū⟩. Note that vowel length is denoted by the absence or presence of a macron (e.g. ⟨A⟩ and ⟨a⟩ map onto short /a/ and ⟨Ā⟩ and ⟨ā⟩ map onto long /aː/).

The Hawaiian conventions for capitalization, numbering, and punctuation are analogous to those in English, except again there is no upper-case variant of ⟨ʻ⟩, so the following vowel is capitalized instead (e.g. *ʻOkakopa* 'October'). In foreign words, such as *kolorofolorokalapona* 'chlorofluorocarbon', one can find the additional consonants: ⟨B b C c D d F f G g J j Q q R r S s T t V v X x Y y Z z⟩.

2.3.2 Modern & missionary orthographies

The primary difference between the missionary and modern Hawaiian orthographies is that the missionary orthography does not encode long vowels or the glottal stop. For example, the following Hawaiian phrases were recorded by a 19th-century German traveller in the missionary orthography: *Ua oia au*, *E ue ae oe ia Ii*, *E ao ae oe ia ia* (Chamisso, 1837, p. 7). In the modern orthography these become: *Ua ʻō ʻia au* 'I am speared', *E uē aʻe ʻoe iā ʻĪʻī* 'You must weep for ʻĪʻī (a person)', and *E aʻo aʻe ʻoe iā ia* 'You teach him' (Elbert and Pukui, 1979, p. 3).

We can convert text in the modern Hawaiian orthography *backward* chronologically to an approximate missionary orthography by mapping each glottal stop ⟨ʻ⟩ to the empty string ϵ, and each long vowel, e.g. ⟨ā ē ī ō ū⟩, to its corresponding short vowel, ⟨a e i o u⟩. As a first approximation, we may treat mappings from the modern-to-missionary orthographies as unambiguously many-to-one; thus there is information loss. We will return to secondary differences between the orthographies in Section 2.6. To illustrate, the following four words in the modern orthography all map to the same missionary string *aa*: *aʻa* (root), *ʻaʻa* (brave), *ʻaʻā* (crumbly lava rock), and *ʻāʻā* (stutter).

The *forward* mapping from missionary-to-modern orthographies is one-to-many. Thus the missionary string *aa* could map to *aʻa*, *ʻaʻa*, *ʻaʻā*, or *ʻāʻā*. The *transliteration problem* we address here seeks to discover how we can use context to recover the information not present in the missionary orthography that modern Hawaiian orthography retains.

Source	Chars	Words
Ulukau(160 texts)	6,518,451	1,334,451
Hi‘iakaikapoliopele	1,272,935	259,947
Wikipedia	577,794	10,221
Total	8,369,180	1,604,619

Table 2.1: Modern data sources and their sizes.

2.3.3 Data sources

We draw on three sources for modern Hawaiian text: the main text of *Hi‘iakaikapoliopele* (Ho‘oulumāhiehie, 2007), 160 short texts from *Ulukau: The Hawaiian Electronic Library*, and the full Hawaiian Wikipedia (see Table 2.1).[1]

For evaluation, we simulate a missionary-era version of the modern texts using the backward mapping described above. In addition, we evaluated our models on a couple of 19th century newspaper samples for which we have parallel missionary-era and modern text. Both simulated and real parallel corpora will be described in Section 2.5.

2.4 Models

We can frame the task of transliterating from missionary-to-modern Hawaiian orthographies as a sequence transduction problem. Many deep learning approaches (e.g. Sutskever et al., 2014; Graves, 2012a) are not easily applicable to this task since we do not have a sufficiently large dataset of parallel texts. Instead, we focus on approaches that mix hand-designed FSTs, used for encoding our expert knowledge, with trained language models including deep learning approaches like RNNLMs (Mikolov et al., 2010a).

As described in the introduction, these methods are similar to those found in hybrid speech recognition with phoneme-level acoustic models, where a language model FST disambiguates the candidates proposed by a pronunciation FST that maps from phonemes to characters or words.

[1] *Ulukau: The Hawaiian Electronic Library*: `http://ulukau.org/`, Hawaiian Wikipedia: `https://haw.wikipedia.org/`. Both accessed 19 May 2018.

2.4.1 End-to-end FSTs

Our initial approach represents the mapping from missionary to modern orthography using a composition of (weighted) FSTs; for a thorough review, see Mohri (1997).

First, we construct a finite state acceptor, I, from the input text. Here we construct a trivial chain-shaped acceptor that accepts only the input text. Each symbol in the input text is represented by a state which emits this symbol on a single transition that moves to the next state. The transition emitting the final symbol in the string leads to the sole accepting state.

Second, we construct an FST called C, which models potential orthography changes that can occur when transliterating from the missionary to modern Hawaiian orthography (see Figure 2.1). For example, two non-deterministic transitions introduce an optional long-vowel map for ⟨a⟩: (a : a) and (a : ā). Another transition inserts glottal stops: (ϵ : ‘). By capturing the orthographic changes we know to occur, the composition $I \circ C$ produces a large set of candidates to be narrowed using the language model. If we were to project this FST to its outputs, we get an weighted acceptor describing a space of candidate modern Hawaiian strings.

Third, we use the modern Hawaiian text from Section 2.3.3 to construct and evaluate a number of character-level n-gram language models, of various combinations of order and Katz backoff and Kneser-Ney (KN) smoothing (Katz, 1987; Kneser and Ney, 1995) (see Section 2.5.1 for a list of models that we trained). N-gram language models can be expressed as weighted FSTs. We denote the n-gram or weighted FST language model as G. Character-level models are used as we wanted to generalize to out-of-vocabulary words, which we expect to occur frequently in a relatively small corpus like the one we have for Hawaiian.

Finally, we use this model to infer modern orthography given a piece of text in missionary orthography as input, then compose the FSTs to form the input-specific *search graph* FST,

$$S = \pi_{out}(I \circ C \circ G),$$

where π_{out} denotes projection to the output symbols of the FST. Note that we can also equivalently interpret S as the intersection of two weighted acceptors:

$$S = \pi_{out}(I \circ C \circ G) = \pi_{out}(I \circ C) \cap G.$$

The minimum cost path through S gives the predicted modern orthography. Of these n-gram-based approaches, we found the Kneser-Ney-based models to

perform best. These approaches will be referred to as FST-C-NGRAM-KN and FST-C_{wb}-NGRAM-KN.

We circumvent the lack of a large, non-simulated parallel corpus by training the language model exclusively on text in the modern Hawaiian orthography. In turn, the orthographic transliteration FST C produces candidates which are disambiguated by the language model. The result is finally evaluated against the ground-truth modern text.

Although the orthographic transliteration model is an approximation, and thus not exhaustive, it embodies an explicit and interpretable representation that can be extended independently of the rest of the model. To illustrate this, we constructed a variant C_{wb} (where wb stands for word boundary). C_{wb} optionally inserts a space after each vowel using an additional arc that maps (ϵ : *space*) (again see Figure 2.1). This variant is able to model some changes in Hawaiian's word-boundary conventions (Wilson, 1976), such as *alaila* becoming *a laila* which demarcates the preposition *a* 'until' and noun *laila* 'then'. We report on the use of C_{wb} to predict modern equivalents of 19th century newspaper samples in Section 2.5. An example prediction can be found in Section 2.6, with more in Appendix A.

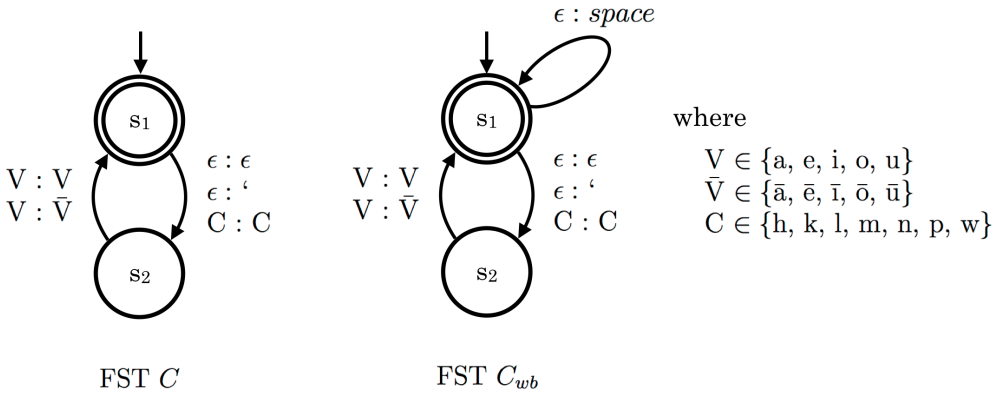


Figure 2.1: Two FSTs. The first, C, transduces between missionary and modern orthographies. The second, C_{wb}, introduces optional spaces (or word boundaries) after a vowel. In each FST, s_1 serves as both the initial and end state, while labelled arrows denote arcs. In the labelled transitions, V and V̄ are variables for short and long vowels, respectively, and C can be any consonant other than ⟨‘⟩. Some arcs, for upper-case letters, numbers, and punctuation, have been omitted for brevity.

2.4.2 Hybrid approach: Composing FSTs and RNNLMs

As an alternative approach, we tried combining the FST C in the previous section with an RNNLM, since RNNLMs often generalize better than n-gram language models (Mikolov et al., 2010a).

An RNN is a neural network that models temporal or sequential data, by iterating a function mapping a state and input to a new state and output. These can be stacked to form a deep RNN. For language modelling, each step of the final RNN layer models a word or character sequence via $p(w_1, \ldots, w_n) = \prod_{i=1}^{n} p(w_i | w_{1:i-1})$ and can be trained by maximum likelihood. Recent language modeling work has typically used the long short-term memory (LSTM) unit, because of its favorable gradient propagation properties (Hochreiter and Schmidhuber, 1997). All RNNs in this chapter are LSTMs.

Our goal in the hybrid approach is to replace the n-gram language model in the end-to-end FST with an RNNLM. While the minimum cost path through an FST can be computed exactly as done in the previous section, it is not straightforward to compose the relation defined by an FST with an arbitrary one like that defined by an RNNLM. Nonetheless, a minimum cost path through the composition of the FST and the RNNLM can be defined as a path (i.e. label sequence) that minimizes the sum of the FST path cost and the RNNLM cost.

We can approximately find a minimum cost path for the composition of the two models by a breadth-first search over the FST graph, using a beam search, as follows. At any particular iteration, consider a single beam element. The beam element holds the current FST and RNN states, and the path taken through the FST so far. We follow each possible arc from the current FST state, each producing a new child beam element, and feed the output symbol into the RNN (unless it is ϵ). We note that there may be duplicate beam elements due the nondeterminicity of the FST, in which case the lower cost edge wins. We sort by the sum of the FST and RNN costs, keep the lowest-cost K, and then proceed to the next iteration. If a beam element is on an accepting state of the FST, it is kept as-is between iterations. Detailed pseudocode is provided in Figure 2.2.

Conceptually, this algorithm performs the same operation as the n-gram language model, except that we replace the n-gram language model with an RNN language model, and then we search over the FST graph, producing scores from FST weights and RNN's outputs. Incidentally, we note that our implementation of the algorithm is slightly different than how we present it in the pseudocode, as we grouped RNN operations into batches for computational efficiency.

$B \leftarrow \{(\text{fst-state} = s_0, \text{rnn-state} = \mathbf{0}, \text{tokens} = [], \text{score} = 0)\}$ ▷ single initial beam element
while $\exists b \in B$: b.fst-state is not final **do**
  $B' = \{\}$ ▷ child beam elements
  for each beam element $b \in B$ that is not in a final state **do**
    for each arc leaving b.fst-state (new state s', output c, weight w) **do**
      ▷ Note that we do not need to use the input symbol for our purposes, but in some settings it may be useful to keep track of it.
      if b.tokens$[-1] \neq \epsilon$ **then**
        $(h', p(y))$ = run RNN one timestep with input (b.rnn-state, c)
        rnncost $= -\log p(b.\text{tokens}[-1])$
          ▷ denotes the negative log probability of the RNN's discrete distribution output at this timestep
      else
        $h' = b$.rnn-state
        rnncost $= 0$
      b' = copy(b)
      b'.score $= b'$.score + rnncost + w
      b'.tokens.append(c)
      b'.rnn-state $= h'$
      $B' = B' \cup \{b'\}$
  $B' = B' \cup \{b \in B : b \text{ is in a final state}\}$
  Sort B' by score, put best (lowest score) K into B

Figure 2.2: Algorithm for approximating the minimum cost path of the composition of an FST with a RNN LM.

2.5 Results

2.5.1 Summary of language models trained

In this chapter, we will refer to the hybrid models as FST-RNNLM in general, and as FST-RNNLM-C and FST-RNNLM-C_{wb} if we want to distinguish which FST was used. The end-to-end FST models will similarly be referred to as FST-C and FST-C_{wb}, with suffixes denoting what kind of n-gram and smoothing were used. For example, FST-C-7GRAM-KN denotes an end-to-end FST with a 7-gram language model and Kneser-Ney smoothing. Here is a list of n-gram language models that we considered (with perplexity scores in parentheses):

- 7-gram Katz backoff (3.15)
- 7-gram Kneser-Ney (3.07)
- 7-gram Kneser-Ney with backoff (3.15)

- 9-gram Katz backoff (3.15)
- **9-gram Kneser-Ney (2.95)**
- 9-gram Kneser-Ney with backoff (3.24)
- 11-gram Katz backoff (3.39)
- **11-gram Kneser-Ney (2.94)**
- 11-gram Kneser-Ney with backoff (3.67)

We note that 13- and higher n-gram language models performed far worse. The numbers in parentheses show character-level perplexities produced using a validation set that was drawn from a synthetic parallel corpus, which will be explained in Section 2.5.3.

We also trained two character-level RNNLMs with the following configurations:

- 2 layers × 200 unit LSTM (2.70)
- **3 layers × 200 unit LSTM (2.65)**

Both RNNLMs were trained with plain SGD, using a batch size of 30, a learning rate of 10, truncated backpropagation through time with 45 unrolling steps, and the gradient renormalized to norm 1 when it exceeded 1. Both RNNLMs used a dropout rate of 0.2 at the input, between RNN layers, and after the last RNN layer. The best n-gram and RNNLMs are highlighted in **bold**. Overall, the best results were produced with the hybrid approach, using RNNLMs.

2.5.2 Evaluation

Because we were unable to find a sufficiently large corpus of parallel texts in the missionary and modern Hawaiian orthographies, we instead trained the n-gram and RNN language models on a corpus of modern Hawaiian texts (*ground-truth*) (see Section 2.3.3). Parallel corpora were only required to test predictions from missionary-era to modern texts, which were produced by composing one of the FSTs, C or C_{wb}, with either an n-gram or RNN language model. To evaluate the accuracy of our approaches, we first derived a synthetic parallel corpus from our collection of modern Hawaiian texts. We also used a small but real parallel corpus, based on two 19th century newspaper texts and their hand-edited modern equivalents. Results based on these parallel corpora are reported in the following subsections.

2.5.3 Simulated parallel corpus (modern texts)

To produce a simulated parallel corpus (*input-missionary*), we systematically reduced the orthography in the modern texts (Section 2.3.3) using the backward mapping described above (Section 2.3.2). We then applied the end-to-end FST and hybrid FST-RNNLM models (Section 2.4), with the aim of learning a forward mapping between orthographies that recovers the lost information.

We evaluated the predicted modern text (*predictions*) by computing

$$\text{CERR} = \frac{d(\text{prediction}, \text{ground-truth})}{d(\text{input-missionary}, \text{ground-truth})},$$

where d denotes character-level edit distance. This is a modification of character error rate, normalized by the distance of the input and target rather than by the length of the target. We note that CERR may be high even when the predictions are very accurate as $d(\text{input-missionary}, \text{ground-truth})$ is small when the text is similar in both orthographies.

Table 2.2 gives more details about the strongest models from both approaches. Out of the Kneser-Ney n-gram models, we found that the FST-C-9GRAM-KN and the version modelling word boundaries (FST-C_{wb}-9GRAM-KN) performed best on the synthetic parallel corpus and newspapers, respectively. C_{wb} was not applied to the synthetic parallel corpus as the synthetic parallel corpus did not model word splitting. However, the hybrid models (FST-RNNLM) outperformed all end-to-end FSTs.

Transliteration model	LM perplexity Valid.	Test	Transliteration performance (%CERR) Corpus	Newspaper 1	Newspaper 2
FST-(C/C_{wb})-9GRAM-KN	2.95	3.02	26.6%	50.7% / 39.3%	52.5% / 47.2%
FST-(C/C_{wb})-11GRAM-KN	2.94	3.02	27.8%	53.9% / 41.3%	54.1% / 48.7%
FST-RNNLM-(C/C_{wb})	**2.65**	**2.69**	**16.3%**	47.2% / **34.3%**	49.8% / **41.2%**

Table 2.2: Performance (%CERR). Slash-separated pairs denote FSTs incapable/capable of inserting word boundaries, respectively; see Section 2.4. The -KN suffix denotes Kneser-Ney smoothing. The data from Section 2.3.3 is used for evaluating the modern-orthography language model perplexity, and "Corpus" evaluates test-set transliteration performance from the synthetic missionary text back to the original modern text.

2.5.4 Real parallel corpus (newspaper texts)

Not content to evaluate the model on simulated missionary orthography, we also evaluated it on two newspaper texts, using selections originally published in 1867 and

1894 for which we had 19th century and manually-edited modern equivalents. The newspaper selections discuss *Kahahana*, one of the last kings of Oʻahu (Kamakau and Perreira, 2002), and *Uluhaimalama*, a garden party and secret political gathering, held after the deposition of Hawaiʻi's last queen (Pukui et al., 2006). Unlike the synthetic missionary corpus evaluation where we did not model word splitting, we found that replacing C with C_{wb} on the newspaper texts significantly improved the output, especially on the FST-RNNLM model. Again, we found a hybrid model (FST-RNNLM-C_{wb}) to be the best performing model overall (Table 2.2).

2.5.5 Predictions on newspaper excerpts

In this section, we show several example results from our model on the two newspaper texts. Further sample predictions on the two texts can be found in Appendix A.

Each block of three lines begins with the input, followed by the prediction, followed by the ground-truth. We denote characters omitted by the model as compared to the ground-truth by *blue and italic*, whereas characters that are erroneously inserted or substituted (or should have been substituted) for another character are marked with red underline. When the incorrect character is a space, the space is replaced with '␣'. Note that correct changes are left unmarked.

Newspaper 1

Input 1: Weheia ka Malapua Alii a Kanuia na Uluwehi no ia Wao.
Prediction 1: Wehe ʻia ka Māla pua Aliʻi a Kanu␣ʻia na Uluwehi nō ia Wao.
Ground-truth 1: Wehe ʻia ka Māla Pua Aliʻi a Kanu ʻia nā Uluwehi no ia Wao.

Input 2: E like no hoi me ka mea i hoike akea ia ae no ka manawa a me ka la e weheia ai a e kanuia ai hoi o na pua a me na mea ulu e ae ma kahi nona ka inoa kilakila maluna ae, pela no i hoea io mai ai i ka Poaha iho la, hora 9 A. M. a mahope mai. Mamua ae o ia manawa, ua lehulehu na poe i pii aku me na mea kanu, maluna o na kaa a malalo no hoi.
Prediction 2: E like nō hoʻi me ka mea i hōʻike ākea ʻia aʻe no ka manawa a me ka lā e wehe ʻia ai a e kanu ʻia ai hoʻi ʻo nā pua a me nā mea ulu ʻē aʻe ma kahi nona ka inoa kilakila ma luna aʻe, pēlā nō i hōʻea ʻiʻo mai ai i ka Poʻahā iho␣la, hora 9 A. M. a ma hope mai. Ma mua aʻe o ia manawa, ua lehulehu nā poʻe i piʻi aku me nā mea kanu, ma luna o nā kaʻa a ma lalo nō hoʻi.
Ground-truth 2: E like nō hoʻi me ka mea i hōʻike ākea ʻia aʻe no ka manawa a me ka lā e wehe ʻia ai a e kanu ʻia ai hoʻi o nā pua a me nā mea ulu ʻē aʻe ma kahi nona ka inoa kilakila ma luna aʻe, pēlā nō i hōʻea ʻiʻo mai ai i ka Pōʻahā ihola, hola 9 A. M. a ma hope mai. Ma mua aʻe o ia manawa, ua lehulehu nā poʻe i piʻi aku me nā mea kanu, ma luna o nā kaʻa a ma

lalo nō hoʻi.

Input 3: Hoomakaia ke Kanu Ana.
Prediction 3: Hoʻomaka ʻia ke Kanu ʻAna.
Ground-truth 3: Hoʻomaka ʻia ke Kanu ʻAna.

Newspaper 2

Input 1: O Kahahana, he alii kapu ia no o Oahu.
Prediction 1: ʻO Kahahana, he aliʻi kapu ia nō o Oʻahu.
Ground-truth 1: ʻO Kahahana, he aliʻi kapu ia no Oʻahu.

Input 2: O Kaionuilalahai ka makuahine, ka moopuna a Kalaniomaiheuila, ke kaikamahine a Kalanikahimakaialii, a laua o Kualu ke kaikuahine, a mua hoi o Kaulahea ka Moi o Maui.
Prediction 2: ʻO Kaʻiʻonuilalahai ka makuahine, ka moʻopuna a Kalaniʻo maʻihe uila, ke kaikamahine a Kalani kahi makaʻi aliʻi, ʻā lāua ʻo Kūʻalu ke kaikuahine, a mua hoʻi o Kaʻula hea, ka Mōʻī o Maui.
Ground-truth 2: ʻO Kaʻionuilalahai ka makuahine, ka moʻopuna a Kalaniʻōmaiheuila, ke kaikamahine a Kalanikahimakaʻialiʻi, a lāua ʻo Kūalu ke kaikuahine, a mua hoʻi o Kaʻulahea, ka Mōʻī o Maui.

Input 3: A o ka makuakane, oia hoi o Elani, no ka ohana a Kupanihi, a o Keopuolani.
Prediction 3: A ʻo ka makua kāne, ʻo ia hoʻi ʻo ʻElani, no ka ʻohana a Kupanihi, a ʻo Keōpūolani.
Ground-truth 3: A ʻo ka makua kāne, ʻo ia hoʻi ʻo ʻĒlani, no ka ʻohana a Kūpānihi, a ʻo Keōpūolani.

2.6 Conclusions and future work

With this chapter we introduced a new transliteration problem to the field, that of mapping between old and new Hawaiian orthographies—where the modern Hawaiian orthography represents linguistic information that is missing from older missionary-era texts. One difficulty of this problem is that there is a limited amount of Hawaiian data, making data-hungry solutions like end-to-end deep learning unlikely to work. To solve the transliteration problem, we therefore proposed two models: the first was implemented end-to-end using weighted FSTs; the second was a hybrid deep learning approach that combined an FST and an RNNLM. Both models gave promising results, but the hybrid approach performed best. It allowed us to use a more powerful recurrent neural network-based language model, despite our dataset's small size.

Factoring a problem like ours into one part that can be modeled exactly using expert domain knowledge and into another part that can be learned directly from data using deep learning is a promising approach for handling low-resource data.

While the role of the RNNLM part of the hybrid approach may be obvious, the FST component plays an important role too. For example, the hand-designed FST can be replaced without needing to re-train the RNNLM. We tried to showcase this modularity by constructing two FSTs which we referred to as C and C_{wb}, where only the latter allowed the insertion of spaces. Future work could extend the FST to model orthographic changes suggested by an error analysis of the current model's predictions.

Input:	Weheia ka Malapua Alii a Kanuia na Uluwehi no ia Wao.
Prediction:	Wehe␣**‘**ia ka M**ā**la␣*p*ua Ali**‘**i a Kanu␣*‘*ia n*a* Uluwehi n**ō** ia Wao.
Ground-truth:	Wehe ‘ia ka Māla Pua Ali‘i a Kanu ‘ia nā Uluwehi no ia Wao.

Table 2.3: Sample prediction. Data from from a 19th century Hawaiian newspaper (Pukui et al., 2006). Correct predictions are **green and bold**. Characters omitted by the model as compared to the ground-truth are denoted by *blue italics*; erroneous insertions or substitutions are denoted by red underline. To make white spaces explicit, we represent them with the symbol '␣'. More sample predictions can be found in Appendix A.

An example of the current model's predictions (i.e. missionary input, predicted modern text, modern ground-truth) is given in Table 2.3. In this example, we see the model correctly predicting some word boundaries, glottal stops and long vowels; however, we note that the model could not predict uppercase *Pua* (correct), because the input text contained lowercase *pua* (incorrect), and no (p : P) transitions were included in C or C_{wb}. Similar observations (see Appendix A) motivate new mappings for consonant substitutions like (r : l) and (s : k) that occur in loanword adaptations (e.g. *rose* ⇒ *loke*). The error analysis also motivates mappings to delete spaces (␣ : ϵ) and to handle contractions, like *na'lii* ⇒ *nā ali‘i*. We could further incorporate linguistic knowledge of Hawaiian into the FST, which tells us, for example, about expected sequences of vowels (Parker Jones, 2010). Additional improvements to the hybrid model might be obtained by increasing the amount of modern Hawaiian text used to train the RNNLM. One way to do this would be to accelerate the rate at which missionary-era Hawaiian texts are modernized. To this end, we hope that the present models will be used within the Hawaiian community to semi-automate, and thereby accelerate, the modernization of old Hawaiian texts.

3

LipNet: visual speech recognition using connectionist temporal classification

Abstract

Lipreading is the task of decoding text from the movement of a speaker's mouth. Traditional approaches separated the problem into two stages: designing or learning visual features, and prediction. More recent deep lipreading approaches are end-to-end trainable (Wand et al., 2016; Chung and Zisserman, 2016b). However, existing work on models trained end-to-end perform only word classification, rather than sentence-level sequence prediction. Studies have shown that human lipreading performance increases for longer words (Easton and Basala, 1982), indicating the importance of features capturing temporal context in an ambiguous communication channel. Motivated by this observation, we present LipNet, a model that maps a variable-length sequence of video frames to text, making use of spatiotemporal convolutions, a recurrent network, and the connectionist temporal classification loss, trained entirely end-to-end. To the best of our knowledge, LipNet is the first end-to-end sentence-level lipreading model that simultaneously learns spatiotemporal visual features and a sequence model. On the GRID corpus, LipNet achieves 95.2% accuracy in sentence-level, overlapped speaker split task, outperforming experienced human lipreaders and the previous 86.4% word-level state-of-the-art accuracy (Gergen et al., 2016).

3.1 Introduction

Lipreading plays a crucial role in human communication and speech understanding, as highlighted by the McGurk effect (McGurk and MacDonald, 1976), where one phoneme's audio dubbed on top of a video of someone speaking a different phoneme results in a third phoneme being perceived.

Lipreading is a notoriously difficult task for humans, specially in the absence of context.[1] Most lipreading actuations, besides the lips and sometimes tongue and teeth, are latent and difficult to disambiguate without context (Fisher, 1968; Woodward and Barber, 1960). For example, Fisher (1968) gives 5 categories of visual phonemes (called *visemes*), out of a list of 23 initial consonant phonemes, that are commonly confused by people when viewing a speaker's mouth. Many of these were asymmetrically confused, and observations were similar for final consonant phonemes.

Consequently, human lipreading performance is poor. Hearing-impaired people achieve an accuracy of only $17 \pm 12\%$ even for a limited subset of 30 monosyllabic words and $21 \pm 11\%$ for 30 compound words (Easton and Basala, 1982). An important goal, therefore, is to automate lipreading. Machine lipreaders have enormous practical potential, with applications in improved hearing aids, silent dictation in public spaces, security, speech recognition in noisy environments, biometric identification, and silent-movie processing.

Machine lipreading is difficult because it requires extracting spatiotemporal features from the video (since both position and motion are important). Recent deep learning approaches attempt to extract those features end-to-end. Most existing work, however, performs only word classification, not sentence-level sequence prediction.

In this chapter, we present LipNet, which is to the best of our knowledge, the first *end-to-end sentence-level* lipreading model. As with modern deep learning based automatic speech recognition (ASR), LipNet is trained end-to-end to make sentence-level predictions. Our model operates at the character-level, using spatiotemporal convolutional neural networks (STCNNs), recurrent neural networks (RNNs), and the connectionist temporal classification loss (CTC) (Graves et al., 2006).

Our empirical results on the GRID corpus (Cooke et al., 2006), one of the few public sentence-level datasets, show that LipNet attains a 95.2% sentence-level word accuracy, in a overlapped speakers split that is popular for benchmarking lipreading methods. The previous best accuracy reported on an aligned word classification

[1] LipNet video: `https://youtube.com/playlist?list=PLXkuFIFnXUAPIrXKgtIpctv2NuSo7xw3k`

version of this task was 86.4% (Gergen et al., 2016). Furthermore, LipNet can generalize across unseen speakers in the GRID corpus with an accuracy of 88.6%.

We also compare the performance of LipNet with that of hearing-impaired people who can lipread on the GRID corpus task. On average, they achieve an accuracy of 52.3%, in contrast to LipNet's 1.69× higher accuracy in the same sentences.

Finally, by applying saliency visualisation techniques (Zeiler and Fergus, 2014; Simonyan et al., 2013), we interpret LipNet's learned behavior, showing that the model attends to phonologically important regions in the video. Furthermore, by computing intra-viseme and inter-viseme confusion matrices at the phoneme level, we show that almost all of LipNet's few erroneous predictions occur within visemes, since context is sometimes insufficient for disambiguation.

3.2 Related Work

In this section, we outline various existing approaches to automated lipreading.

Automated lipreading Most existing work on lipreading does not employ deep learning. Such work requires either heavy preprocessing of frames to extract image features, temporal preprocessing of frames to extract video features (e.g., optical flow or movement detection), or other types of handcrafted vision pipelines (Matthews et al., 2002; Zhao et al., 2009; Gurban and Thiran, 2009; Papandreou et al., 2007; Papandreou et al., 2009; Pitsikalis et al., 2006; Lucey and Sridharan, 2006; Papandreou et al., 2009). The automated lipreading literature is too vast to adequately cover, so we refer the reader to Zhou et al. (2014) for an extensive review.

Notably, Goldschen et al. (1997) were the first to do visual-only sentence-level lipreading using hidden Markov models (HMMs) in a limited dataset, using hand-segmented phones. Later, Neti et al. (2000) were the first to do sentence-level audiovisual speech recognition using an HMM combined with hand-engineered features, on the IBM ViaVoice (Neti et al., 2000) dataset. The authors improve speech recognition performance in noisy environments by fusing visual features with audio ones. The dataset contains 17111 utterances of 261 speakers for training (about 34.9 hours) and is not publicly available. As stated, their visual-only results cannot be interpreted as visual-only recognition, as they are used as rescoring of the noisy audio-only lattices. Using a similar approach, Potamianos et al. (2003) report speaker independent and speaker adapted 91.62%, 82.31% WER in the same dataset respectively, and 38.53%, 16.77% WER in the connected DIGIT corpus, which contains sentences of digits.

Furthermore, Gergen et al. (2016) use speaker-dependent training on an LDA-transformed version of the Discrete Cosine Transforms of the mouth regions in an HMM/GMM system. This work holds the previous state-of-the-art on the GRID corpus with a speaker-dependent accuracy of 86.4%. Generalisation across speakers and extraction of motion features is considered an open problem, as noted in (Zhou et al., 2014). LipNet addresses both of these issues.

Classification with deep learning In recent years, there have been several attempts to apply deep learning to lipreading. However, all of these approaches perform only word or phoneme classification, whereas LipNet performs full sentence sequence prediction. Approaches include learning multimodal audio-visual representations (Ngiam et al., 2011; Sui et al., 2015; Ninomiya et al., 2015; Petridis and Pantic, 2016), learning visual features as part of a traditional speech-style processing pipeline (e.g. HMMs, GMM-HMMs, etc.) for classifying words and/or phonemes (Almajai et al., 2016; Takashima et al., 2016; Noda et al., 2014; Koller et al., 2015), or combinations thereof (Takashima et al., 2016). Many of these approaches mirror early progress in applying neural networks for acoustic processing in speech recognition (Hinton et al., 2012).

Chung and Zisserman (2016b) propose spatial and spatiotemporal convolutional neural networks, based on VGG, for word classification. The architectures are evaluated on a word-level dataset BBC TV (333 and 500 classes), but, as reported, their spatiotemporal models fall short of the spatial architectures by an average of around 14%. Additionally, their models cannot handle variable sequence lengths and they do not attempt sentence-level sequence prediction.

Chung and Zisserman (2016a) train an audio-visual max-margin matching model for learning pre-trained mouth features, which they use as inputs to an LSTM for 10-phrase classification on the OuluVS2 dataset, as well as a non-lipreading task.

Wand et al. (2016) introduce LSTM recurrent neural networks for lipreading but address neither sentence-level sequence prediction nor speaker independence.

Garg et al. (2016) apply a VGG pre-trained on faces to classifying words and phrases from the MIRACL-VC1 dataset, which has only 10 words and 10 phrases. However, their best recurrent model is trained by freezing the VGGNet parameters and then training the RNN, rather than training them jointly. Their best model achieves only 56.0% word classification accuracy, and 44.5% phrase classification accuracy, despite both of these being 10-class classification tasks.

Sequence prediction in speech recognition The field of automatic speech recognition (ASR) would not be in the state it is today without modern advances in deep learning, many of which have occurred in the context of ASR (Graves

et al., 2006; Dahl et al., 2012; Hinton et al., 2012). The connectionist temporal classification loss (CTC) of Graves et al. (2006) drove the movement from deep learning as a component of ASR, to deep ASR systems trained end-to-end (Graves and Jaitly, 2014; Maas et al., 2015; Amodei et al., 2016). As mentioned earlier, much recent lipreading progress has mirrored early progress in ASR, but stopping short of sequence prediction.

LipNet is the first end-to-end model that performs sentence-level sequence prediction for visual speech recognition. That is, we demonstrate the first work that takes as input as sequence of images and outputs a distribution over sequences of tokens; it is trained end-to-end using CTC and thus also does not require alignments.

Lipreading Datasets Lipreading datasets (AVICar, AVLetters, AVLetters2, BBC TV, CUAVE, OuluVS1, OuluVS2) are plentiful (Zhou et al., 2014; Chung and Zisserman, 2016b), but most only contain single words or are too small. One exception is the GRID corpus (Cooke et al., 2006), which has audio and video recordings of 34 speakers who produced 1000 sentences each, for a total of 28 hours across 34000 sentences. Table 3.1 summarises state-of-the-art performance in each of the main lipreading datasets.

Table 3.1: Existing lipreading datasets and the state-of-the-art accuracy reported on these. The size column represents the number of utterances used by the authors for training. Although the GRID corpus contains entire sentences, Gergen et al. (2016) consider only the simpler case of predicting isolated words. LipNet predicts sequences and hence can exploit temporal context to attain much higher accuracy. Phrase-level approaches were treated as plain classification.

Method	Dataset	Size	Output	Accuracy
Fu et al. (2008)	AVICAR	851	Digits	37.9%
Hu, Li, et al. (2016)	AVLetters	78	Alphabet	64.6%
Papandreou et al. (2009)	CUAVE	1800	Digits	83.0%
Chung and Zisserman (2016b)	OuluVS1	200	Phrases	91.4%
Chung and Zisserman (2016a)	OuluVS2	520	Phrases	94.1%
Chung and Zisserman (2016b)	BBC TV	>400000	Words	65.4%
Gergen et al. (2016)	GRID	29700	Words*	86.4%
LipNet	GRID	28775	**Sentences**	**95.2**%

We use the GRID corpus to evaluate LipNet because it is sentence-level and has the most data. The sentences are drawn from the following simple grammar: $command^{(4)} + color^{(4)} + preposition^{(4)} + letter^{(25)} + digit^{(10)} + adverb^{(4)}$, where the number denotes how many word choices there are for each of the 6 word categories.

The categories consist of, respectively, {bin, lay, place, set}, {blue, green, red, white}, {at, by, in, with}, $\{A, \ldots, Z\} \backslash \{W\}$, {zero, ..., nine}, and {again, now, please, soon}, yielding 64000 possible sentences. For example, two sentences in the data are "set blue by A four please" and "place red at C zero again".

3.3 LipNet

LipNet is a neural network architecture for lipreading that maps variable-length sequences of video frames to text sequences, and is trained end-to-end. In this section, we describe LipNet's building blocks and architecture.

3.3.1 Gated Recurrent Unit

As introduced in the background section, Gated Recurrent Unit (GRU) (Cho et al., 2014; Chung et al., 2014) is a type of recurrent neural network (RNN) that improves upon earlier RNNs by adding cells and gates for propagating information over more time-steps and learning to control this information flow. It is similar to the Long Short-Term Memory (LSTM) RNN (Hochreiter and Schmidhuber, 1997). We use the standard formulation:

$$
\begin{aligned}
[\mathbf{u}_t, \mathbf{r}_t]^\top &= \text{sigm}(\mathbf{W}_z \mathbf{z}_t + \mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{b}_g) \\
\tilde{\mathbf{h}}_t &= \tanh(\mathbf{U}_z \mathbf{z}_t + \mathbf{U}_h (\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_h) \\
\mathbf{h}_t &= (\mathbf{1} - \mathbf{u}_t) \odot \mathbf{h}_{t-1} + \mathbf{u}_t \odot \tilde{\mathbf{h}}_t
\end{aligned}
$$

where $\mathbf{z} := \{\mathbf{z}_1, \ldots, \mathbf{z}_T\}$ is the input sequence to the RNN, $\odot$ denotes element-wise multiplication, and $\text{sigm}(r) = 1/(1 + \exp(-r))$.

We use a bidirectional GRU (Bi-GRU) as introduced by Graves and Schmidhuber (2005) in the context of LSTMs: one RNN maps $\{\mathbf{z}_1, \ldots, \mathbf{z}_T\} \mapsto \{\overrightarrow{\mathbf{h}_1}, \ldots, \overrightarrow{\mathbf{h}_T}\}$, and another $\{\mathbf{z}_T, \ldots, \mathbf{z}_1\} \mapsto \{\overleftarrow{\mathbf{h}_1}, \ldots, \overleftarrow{\mathbf{h}_T}\}$, then $\mathbf{h}_t := [\overrightarrow{\mathbf{h}_t}, \overleftarrow{\mathbf{h}_t}]$. The Bi-GRU ensures that $\mathbf{h}_t$ depends on $\mathbf{z}_{t'}$ for all t'. To parameterize a distribution over sequences, at time-step t let $p(u_t|\mathbf{z}) = \text{softmax}(\text{mlp}(\mathbf{h}_t; \mathbf{W}_{mlp}))$, where mlp is a feed-forward network with weights $\mathbf{W}_{mlp}$. Then we can define the distribution over length-T sequences as $p(u_1, \ldots, u_T|\mathbf{z}) = \prod_{1 \le t \le T} p(u_t|\mathbf{z})$, where T is determined by $\mathbf{z}$, the input to the GRU. In LipNet, $\mathbf{z}$ is the output of the STCNN.

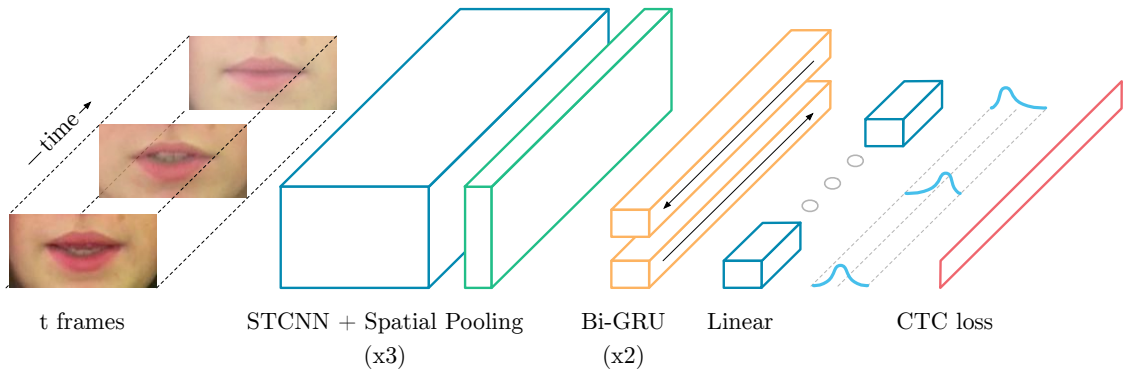


Figure 3.1: LipNet architecture. A sequence of T frames is used as input, and is processed by 3 layers of STCNN, each followed by a spatial max-pooling layer. The features extracted are processed by 2 Bi-GRUs; each time-step of the GRU output is processed by a linear layer and a softmax. This end-to-end model is trained with CTC.

3.3.2 Connectionist Temporal Classification

The connectionist temporal classification (CTC) loss (Graves et al., 2006) is widely used in modern speech recognition as it eliminates the need for training data that aligns inputs to target outputs (Amodei et al., 2016; Graves and Jaitly, 2014; Maas et al., 2015). Given a model that outputs a sequence of discrete distributions over the token classes (vocabulary) augmented with a special "blank" token, CTC computes the probability of a sequence by marginalising over all sequences that are defined as equivalent to this sequence. This simultaneously removes the need for alignments and addresses variable-length sequences. Let V denote the set of tokens that the model classifies at a single time-step of its output (vocabulary), and the blank-augmented vocabulary $\tilde{V} = V \cup \{\textvisiblespace\}$ where $\textvisiblespace$ denotes the CTC blank symbol. Define the function $\mathcal{B} : \tilde{V}^* \to V^*$ that, given a string over $\tilde{V}$, deletes adjacent duplicate characters and removes blank tokens. For a label sequence $y \in V^*$, CTC defines

$$p(y|\mathbf{x}) = \sum_{u \in \mathcal{B}^{-1}(y) \text{ s.t. } |u|=T} p(u_1, \ldots, u_T|\mathbf{x}),$$

where T is the number of time-steps in the sequence model.

For example, if $T = 3$, CTC defines the probability of a string "am" as $p(aam) + p(amm) + p(\textvisiblespace am) + p(a\textvisiblespace m) + p(am\textvisiblespace)$. This sum is computed efficiently by dynamic programming, allowing us to perform maximum likelihood.

3.3.3 LipNet Architecture

The videos were processed with DLib face detector (King, 2009) and the iBug face shape predictor with 68 landmarks (Sagonas et al., 2013). The RGB input

frames were normalized per-channel by the following means and standard deviations estimated on a subset of the training set: $\mu_R = 0.7136$,$\sigma_R = 0.1138$,$\mu_G = 0.4906$,$\sigma_G = 0.1078$,$\mu_B = 0.3283$,$\sigma_B = 0.0917$.

Figure 3.1 illustrates the LipNet architecture, which starts with 3×(spatiotemporal convolutions, channel-wise dropout, spatial max-pooling). Subsequently, the features extracted are followed by two Bi-GRUs. The Bi-GRUs are crucial for efficient further aggregation of the STCNN output. Finally, a linear transformation is applied at each time-step, followed by a softmax over the vocabulary augmented with the CTC blank, and then the CTC loss. All layers use rectified linear unit (ReLU) activation functions.

Table 3.2 summarises the LipNet architecture hyperparameters, where T denotes time, C denotes channels, F denotes feature dimension, H and W denote height and width and V denotes the number of words in the vocabulary including the CTC blank symbol.

Table 3.2: LipNet architecture hyperparameters. Note that spatiotemporal convolution sizes depend on the number of channels, and the kernel's three dimensions. Spatiotemporal kernel sizes are specified in the same order as the input size dimensions. The input dimension orderings are given in parentheses in the input size column. Layers after the Bi-GRU are applied per-timestep.

Layer	Size / Stride / Pad	Input size	Dimension order
STCNN	$3 \times 5 \times 5$ / 1,2,2 / 1,2,2	$75 \times 3 \times 50 \times 100$	$T \times C \times H \times W$
Pool	$1 \times 2 \times 2$ / 1,2,2	$75 \times 32 \times 25 \times 50$	$T \times C \times H \times W$
STCNN	$3 \times 5 \times 5$ / 1,2,2 / 1,2,2	$75 \times 32 \times 12 \times 25$	$T \times C \times H \times W$
Pool	$1 \times 2 \times 2$ / 1,2,2	$75 \times 64 \times 12 \times 25$	$T \times C \times H \times W$
STCNN	$3 \times 3 \times 3$ / 1,2,2 / 1,1,1	$75 \times 64 \times 6 \times 12$	$T \times C \times H \times W$
Pool	$1 \times 2 \times 2$ / 1,2,2	$75 \times 96 \times 6 \times 12$	$T \times C \times H \times W$
Bi-GRU	256	$75 \times (96 \times 3 \times 6)$	$T \times (C \times H \times W)$
Bi-GRU	256	75×512	$T \times F$
Linear	$27 +$ blank	75×512	$T \times F$
Softmax		75×28	$T \times V$

3.3.4 Baseline-LSTM Architecture

Baseline-LSTM replicates the setup of Wand et al. (2016), and is trained the same way as LipNet. The model uses two LSTM layers with 128 neurons. The input frames were converted to grayscale and were down-sampled to 50×25px,

dropout was turned off ($p = 0$), and the parameters were initialised uniformly with values between [−0.05,0.05].

3.3.5 Implementation

LipNet is implemented using Lua Torch alongside the warp-ctc CTC library (Amodei et al., 2016), and makes use of Stanford-CTC's decoder implementation. The network parameters were initialised using He initialisation (He et al., 2015), apart from the square GRU matrices that were orthogonally initialised, as described in (Chung et al., 2014). The models were trained with channel-wise dropout (dropout rate $p = 0.5$) after each pooling layer and mini-batches of size 50. We used the optimiser Adam (Kingma and Ba, 2014) with a learning rate of 10^{-4}, and the default hyperparameters: a first-moment momentum coefficient of 0.9, a second-moment momentum coefficient of 0.999, and the numerical stability parameter $\epsilon = 10^{-8}$.

The CER and WER were computed using predictions produced using CTC beam search with the following parameters for Stanford-CTC's decoder implementation: beam width 200, $\alpha = 1$, and $\beta = 1.5$. As CTC has conditionally independent predictions at each timestep, we use a character 5-gram binarized language model at test time.

3.4 Lipreading Evaluation

In this section, we evaluate LipNet on the GRID corpus. The augmentation methods employed don't make use of external data and rely purely on the GRID corpus.

3.4.1 Data Augmentation

Preprocessing The GRID corpus consists of 34 subjects, each narrating 1000 sentences. The videos for speaker 21 are missing, and a few others are empty or corrupt, leaving 32746 usable videos. We employ a split (unseen speakers; not previously used in the literature) holding out the data of two male speakers (1 and 2) and two female speakers (20 and 22) for evaluation (3971 videos). The remainder is used for training (28775 videos). We also use a sentence-level variant of the split (overlapped speakers) similar to Wand et al. (2016), where 255 random sentences from each speaker are used for evaluation. All remaining data from all speakers is pooled together for training. All videos are 3 seconds long with a frame rate of 25fps. The videos were processed with the DLib face detector and the iBug face landmark predictor (Sagonas et al., 2013) producing 68 landmarks, coupled with

an online Kalman filter. Using these landmarks, we apply an affine transformation to extract a mouth-centred crop of size 100×50 pixels per frame. We standardize the RGB channels over the whole training set to have zero mean and unit variance.

Augmentation We augment the dataset with simple transformations to reduce overfitting. First, we train on both the regular and the horizontally mirrored image sequence. Second, since the dataset provides word start and end timings for each sentence video, we augment the sentence-level training data with video clips of individual words as additional training instances. These instances have a decay rate of 0.925. Third, to encourage resilience to varying motion speeds by deletion and duplication of frames, this is performed with a per-frame probability of 0.05. The same augmentation methods were followed in all proposed baselines and models.

3.4.2 Baselines

To evaluate LipNet, we compare its performance to that of three hearing-impaired people who can lipread, as well as three ablation models inspired by recent state-of-the-art work (Chung and Zisserman, 2016b; Wand et al., 2016).

Hearing-Impaired People This baseline was performed by three members of the Oxford Students' Disability Community. After being introduced to the grammar of the GRID corpus, they observed 10 minutes of annotated videos from the training dataset, then annotated 300 random videos from the evaluation dataset. When uncertain, they were asked to pick the most probable answer.

Baseline-LSTM Using the sentence-level training setup of LipNet, we replicate the model architecture of the previous deep learning GRID corpus state-of-the-art (Wand et al., 2016). See Section 3.3.4 for implementation details.

Baseline-2D Based on the LipNet architecture, we replace the STCNN with spatial-only convolutions similar to those of Chung and Zisserman (2016b). Notably, contrary to the results we observe with LipNet, Chung and Zisserman (2016b) report 14% and 31% poorer performance of their STCNNs compared to the 2D architectures in their two datasets.

Baseline-NoLM Identical to LipNet, but with the language model used in beam search disabled.

3.4.3 Performance Evaluation

To measure the performance of LipNet and the baselines, we compute the word error rate (WER) and the character error rate (CER), standard metrics for the performance of ASR models. We produce approximate maximum-probability

predictions from LipNet by performing CTC beam search. WER (or CER) is defined as the minimum number of word (or character) insertions, substitutions, and deletions required to transform the prediction into the ground truth, divided by the number of words (or characters) in the ground truth. Note that WER is usually equal to classification error when the predicted sentence has the same number of words as the ground truth, particularly in our case since almost all errors are substitution errors.

Table 3.3 summarises the performance of LipNet compared to the baselines. According to the literature, the accuracy of human lipreaders is around 20% (Easton and Basala, 1982; Hilder et al., 2009). As expected, the fixed sentence structure and the limited subset of words for each position in the GRID corpus facilitate the use of context, increasing performance. On the unseen speakers split, the three hearing-impaired people achieve 57.3%, 50.4%, and 35.5% WER respectively, yielding an average of 47.7% WER.

Table 3.3: Performance of LipNet on the GRID dataset compared to the baselines, measured on two splits: (a) evaluating on only unseen speakers, and (b) evaluating on a 255 video subset of each speakers' sentences.

	Unseen Speakers		**Overlapped Speakers**	
Method	**CER**	**WER**	**CER**	**WER**
Hearing-Impaired Person (avg)	–	47.7%	–	–
Baseline-LSTM	38.4%	52.8%	15.2%	26.3%
Baseline-2D	16.2%	26.7%	4.3%	11.6%
Baseline-NoLM	6.7%	13.6%	2.0%	5.6%
LipNet	**6.4%**	**11.4%**	**1.9%**	**4.8%**

For both unseen and overlapped speakers evaluation, the highest performance is achieved by the architectures enhanced with convolutional stacks. LipNet exhibits a 2.3× higher performance in the overlapped compared to the unseen speakers split. For unseen speakers, Baseline-2D and LipNet achieve 1.8× and 4.2× lower WER, respectively, than hearing-impaired people.

The WER for unseen speakers Baseline-2D is 26.7%, whereas for LipNet it is 2.3× lower, at 11.4%. Similarly, the error rate for overlapped speakers was 2.4× lower for LipNet compared to Baseline-2D. Both results demonstrate the importance of combining STCNNs with RNNs. This performance difference confirms the intuition that extracting spatiotemporal features using a STCNN is better than aggregating spatial-only features. This observation contrasts with the empirical observations

of Chung and Zisserman (2016b). Furthermore, LipNet's use of STCNN, RNNs, and CTC cleanly allow processing both variable-length input and variable-length output sequences, whereas the architectures of Chung and Zisserman (2016b) and Chung and Zisserman (2016a) only handle the former.

Baseline-LSTM exhibits the lowest performance, in both unseen and overlapped speakers, with 52.8% and 26.3% WER, respectively. Interestingly, although Baseline-LSTM replicates the architecture of Wand et al. (2016), and despite the numerous data augmentation methods, the model performs 1.3× lower than the reported 79.6% word-level accuracy illustrating the difficulty of a sentence-level task even in a restricted grammar.

Finally, by disabling the language model, the Baseline-NoLM exhibits approximately 1.2× higher WER than our proposed model.

3.4.4 Learned representations

In this section, we analyse the learned representations of LipNet from a phonological perspective. First, we create saliency visualisations (Simonyan et al., 2013; Zeiler and Fergus, 2014) to illustrate where LipNet has learned to attend. In particular, we feed an input into the model and greedily decode an output sequence, yielding a CTC alignment $\hat{u} \in \tilde{V}^*$ (following the notation of Sections 3.3.1 and 3.3.2). Then, we compute the gradient of $\sum_t p(\hat{u}_t|\mathbf{x})$ with respect to the input video frame sequence, but unlike Simonyan et al. (2013), we use guided backpropagation (Springenberg et al., 2014). Second, we train LipNet to predict ARPAbet phonemes, instead of characters, to analyse visual phoneme similarities using intra-viseme and inter-viseme confusion matrices.

Saliency Maps

We apply saliency visualisation techniques to interpret LipNet's learned behavior, showing that the model attends to phonologically important regions in the video. In particular, in Figure 3.2 we analyse two saliency visualisations for the words *please* and *lay* for speaker 25, based on Ashby (2013).

The production of the word *please* requires a great deal of articulatory movement at the beginning: the lips are pressed firmly together for the bilabial plosive /p/ (frame 1). At the same time, the blade of the tongue comes in contact with the alveolar ridge in anticipation of the following lateral /l/. The lips then part, allowing the compressed air to escape between the lips (frame 2). The jaw and lips then open further, seen in the distance between the midpoints of the upper and lower

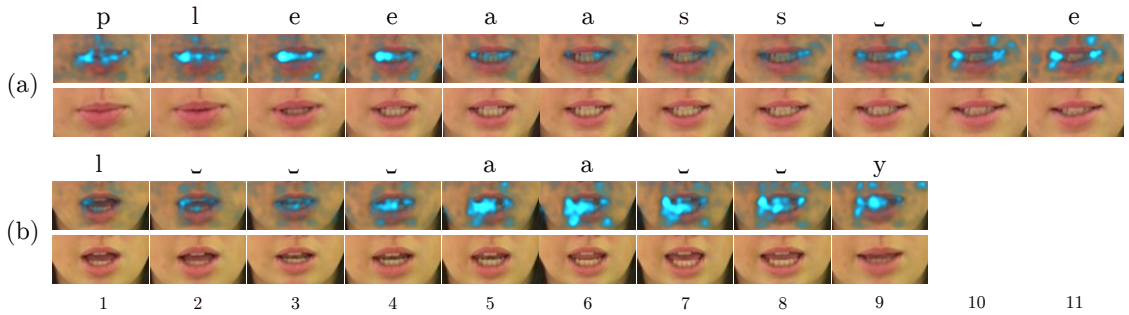


Figure 3.2: Saliency maps for the words (a) *please* and (b) *lay*, produced by backpropagation to the input, showing the places where LipNet has learned to attend. The pictured transcription is given by greedy CTC decoding. CTC blanks are denoted by '␣'.

lips, and the lips spread (increasing the distance between the corners of the mouth), for the close vowel /iy/ (frame 3–4). Since this is a relatively steady-state vowel, lip position remains unchanged for the rest of its duration (frames 4–8), where the attention level drops considerably. The jaw and the lips then close slightly, as the blade of the tongue needs to be brought close to the alveolar ridge, for /z/ (frames 9–10), where attention resumes.

Lay is interesting since the bulk of frontally visible articulatory movement involves the blade of the tongue coming into contact with the alveolar ridge for /l/ (frames 2–6), and then going down for the vowel /ey/ (frames 7–9). That is exactly where most of LipNet's attention is focused, as there is little change in lip position.

Visemes

According to DeLand (1931) and Fisher (1968), Alexander Graham Bell first hypothesised that multiple phonemes may be visually identical on a given speaker. This was later verified, giving rise to the concept of a *viseme*, a visual equivalent of a phoneme (Woodward and Barber, 1960; Fisher, 1968). For our analysis, we use the phoneme-to-viseme mapping of Neti et al. (2000), clustering the phonemes into the following categories: Lip-rounding based vowels (V), Alveolar-semivowels (A), Alveolar-fricatives (B), Alveolar (C), Palato-alveolar (D), Bilabial (E), Dental (F), Labio-dental (G), and Velar (H). The full mapping can be found in Table 3.4. Table 3.4 shows the phoneme to viseme clustering of Neti et al. (2000) and Figure 3.3 shows LipNet's full phoneme confusion matrix. The GRID corpus contain 31 out of the 39 phonemes in ARPAbet.

We compute confusion matrices between phonemes and then group phonemes into viseme clusters, following Neti et al. (2000). Figure 3.4 shows the confusion

Table 3.4: Phoneme to viseme clustering of Neti et al. (2000).

Code	Viseme Class	Phonemes in Cluster
V1		/ao/ /ah/ /aa/ /er/ /oy/ /aw/ /hh/
V2	Lip-rounding based vowels	/uw/ /uh/ /ow/
V3		/ae/ /eh/ /ey/ /ay/
V4		/ih/ /iy/ /ax/
A	Alveolar-semivowels	/l/ /el/ /r/ /y/
B	Alveolar-fricatives	/s/ /z/
C	Alveolar	/t/ /d/ /n/ /en/
D	Palato-alveolar	/sh/ /zh/ /ch/ /jh/
E	Bilabial	/p/ /b/ /m/
F	Dental	/th/ /dh/
G	Labio-dental	/f/ /v/
H	Velar	/ng/ /k/ /g/ /w/
S	Silence	/sil/ /sp/

matrices of the 3 most confused viseme categories, as well as the confusions between the viseme categories. The full phoneme confusion matrix is shown in Figure 3.3.

Given that the speakers are British, the confusion between /aa/ and /ay/ (Figure 3.4a) is most probably due to the fact that the first element, and the greater part, of the diphthong /ay/ is articulatorily identical with /aa/: an open back unrounded vowel (Ferragne and Pellegrino, 2010). The confusion of /ih/ (a rather close vowel) and /ae/ (a very open vowel) is at first glance surprising, but in fact in the sample /ae/ occurs only in the word *at*, which is a function word normally pronounced with a reduced, weak vowel /ah/. /ah/ and /ih/ are the most frequent unstressed vowels and there is a good deal of variation within and between them, e.g. *priv<u>a</u>te* and *watch<u>e</u>s* (Cruttenden, 2014).

The confusion within the categories of bilabial stops /p b m/ and alveolar stops /t d n/ (Figures 3.4b-c) is unsurprising: complete closure at the same place of articulation makes them look practically identical. The differences of velum action and vocal fold vibration are unobservable from the front.

Finally, the quality of the viseme categorisation of Neti et al. (2000) is confirmed by the fact that the matrix in Figure 3.4d is diagonal, with only minor confusion between alveolar (C) and palato-alveolar (D) visemes. Articulatorily, alveolar /s z/ and palato-alveolar /sh zh/ fricatives are distinguished by only a small difference in tongue position: against the palate just behind the alveolar ridge, which is not easily observed from the front. The same can be said about dental /th/ and alveolar /t/.

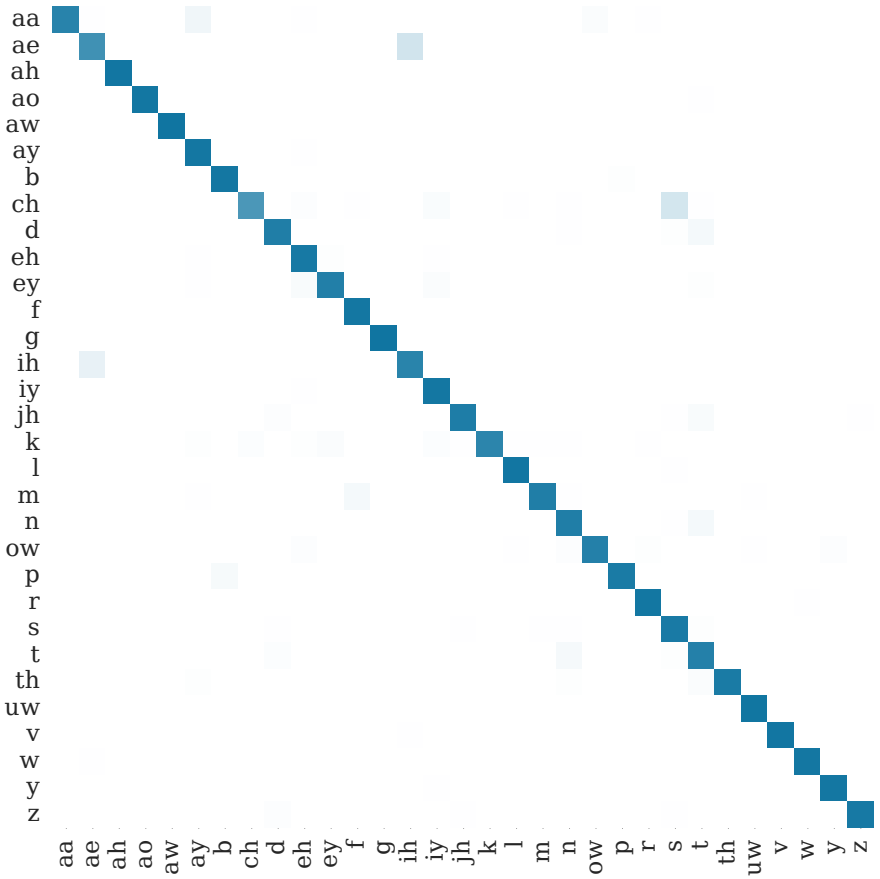
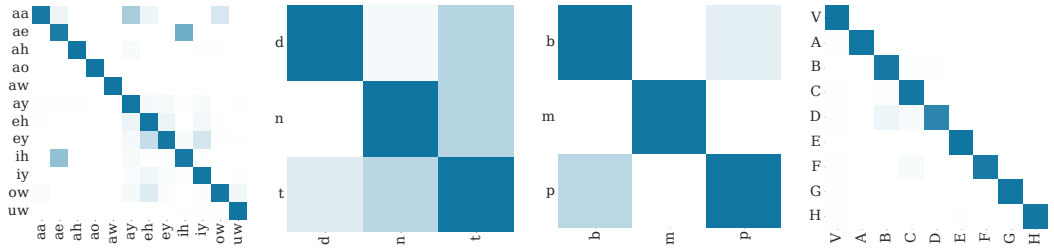


Figure 3.3: LipNet's full phoneme confusion matrix.



(a) Lip-rounding vowels **(b)** Alveolar **(c)** Bilabial **(d)** Viseme Categories

Figure 3.4: Intra-viseme and inter-viseme confusion matrices, depicting the three categories with the most confusions, as well as the confusions between viseme clusters. Colors are row-normalised to emphasize the errors.

3.5 Conclusions

We proposed LipNet, the first model to apply deep learning to end-to-end learning of a model that maps sequences of image frames of a speaker's mouth to entire sentences. The end-to-end model eliminates the need to segment videos into words before predicting a sentence. LipNet requires neither hand-engineered spatiotemporal

visual features nor a separately-trained sequence model.

Our empirical evaluation illustrates the importance of spatiotemporal feature extraction and efficient temporal aggregation, confirming the intuition of Easton and Basala (1982). Furthermore, LipNet greatly outperforms a human lipreading baseline, exhibiting 4.1× better performance, and 4.8% WER which is 2.8× lower than the word-level state-of-the-art (Gergen et al., 2016) in the GRID corpus.

While LipNet is already an empirical success, the deep speech recognition literature (Amodei et al., 2016) suggests that performance will only improve with more data. In future work, we hope to demonstrate this by applying LipNet to larger datasets, such as a sentence-level variant of that collected by Chung and Zisserman (2016b).

Some applications, such as silent dictation, demand the use of video only. However, to extend the range of potential applications of LipNet, we aim to apply this approach to a jointly trained audio-visual speech recognition model, where visual input assists with robustness in noisy environments.

4

Large-scale visual speech recognition

Abstract

This work presents a scalable solution to continuous visual speech recognition. To achieve this, we constructed the largest existing visual speech recognition dataset, consisting of pairs of text and video clips of faces speaking (3,886 hours of video). In tandem, we designed and trained an integrated lipreading system, consisting of a video processing pipeline that maps raw video to stable videos of lips and sequences of phonemes, a scalable deep neural network that maps the lip videos to sequences of phoneme distributions, and a phoneme-to-word speech decoder that outputs sequences of words. The proposed system achieves a word error rate (WER) of 40.9% as measured on a held-out set. In comparison, professional lipreaders achieve either 86.4% or 92.9% WER on the same dataset when having access to additional types of contextual information. Our approach significantly improves on previous lipreading approaches, including variants of *LipNet* and of *Watch, Attend, and Spell* (WAS), which are only capable of 89.8% and 76.8% WER respectively.

4.1 Introduction and motivation

Deep learning techniques have allowed for significant advances in lipreading over the last few years (Assael* et al., 2016; Chung et al., 2017; Thanda and Venkatesan, 2017; Koumparoulis et al., 2017; Chung and Zisserman, 2017; Xu et al., 2018). However, these approaches have often been limited to narrow vocabularies, and relatively small datasets (Assael* et al., 2016; Thanda and Venkatesan, 2017; Xu et al., 2018). Often the approaches focus on single-word classification (Hinton et al., 2012; Chung and Zisserman, 2016b; Wand et al., 2016; Stafylakis and

Tzimiropoulos, 2017; Ngiam et al., 2011; Sui et al., 2015; Ninomiya et al., 2015; Petridis and Pantic, 2016; Petridis et al., 2017; Noda et al., 2014; Koller et al., 2015; Almajai et al., 2016; Takashima et al., 2016; Wand and Schmidhuber, 2017) and do not attack the continuous recognition setting. In this chapter, we contribute a novel method for large-vocabulary continuous visual speech recognition. We report substantial reductions in word error rate (WER) over the state-of-the-art approaches even with a larger vocabulary.

Assisting people with speech impairments is a key motivating factor behind this work. Visual speech recognition could positively impact the lives of hundreds of thousands of patients with speech impairments worldwide. For example, in the U.S. alone 103,925 tracheostomies were performed in 2014 (HCUPnet, 2014), a procedure that can result in a difficulty to speak (disphonia) or an inability to produce voiced sound (aphonia). While this paper focuses on a scalable solution to lipreading using a vast diverse dataset, we also expand on this important medical application in Section 4.2. The discussion there has been provided by medical experts and is aimed at medical practitioners.

We propose a novel lipreading system, illustrated in Figure 4.1, which transforms raw video into a word sequence. The first component of this system is a data processing pipeline used to create the *Large-Scale Visual Speech Recognition* (LSVSR) dataset used in this work, distilled from YouTube videos and consisting of phoneme sequences paired with video clips of faces speaking (3,886 hours of video). The creation of the dataset alone required a non-trivial combination of computer vision and machine learning techniques. At a high-level this process takes as input raw video and annotated audio segments, filters and preprocesses them, and produces a collection of aligned phoneme and lip frame sequences. Compared to previous work on visual speech recognition, our pipeline uses landmark smoothing, a blurriness filter, an improved speaking classifier network and outputs phonemes. The details of this process are described in Section 4.4.

Next, this work introduces a new neural network architecture for lipreading, which we call *Vision to Phoneme* (V2P), trained to produce a sequence of phoneme distributions given a sequence of video frames. In light of the large scale of our dataset, the network design has been highly tuned to maximize predictive performance subject to the strong computational and memory limits of modern GPUs in a distributed setting. In this setting we found that techniques such as group normalization (Wu and He, 2018) to be key to the reported results. Furthermore, our approach is the first to combine a deep learning-based visual speech recognition model with phoneme-to-word decoding techniques. By decoupling

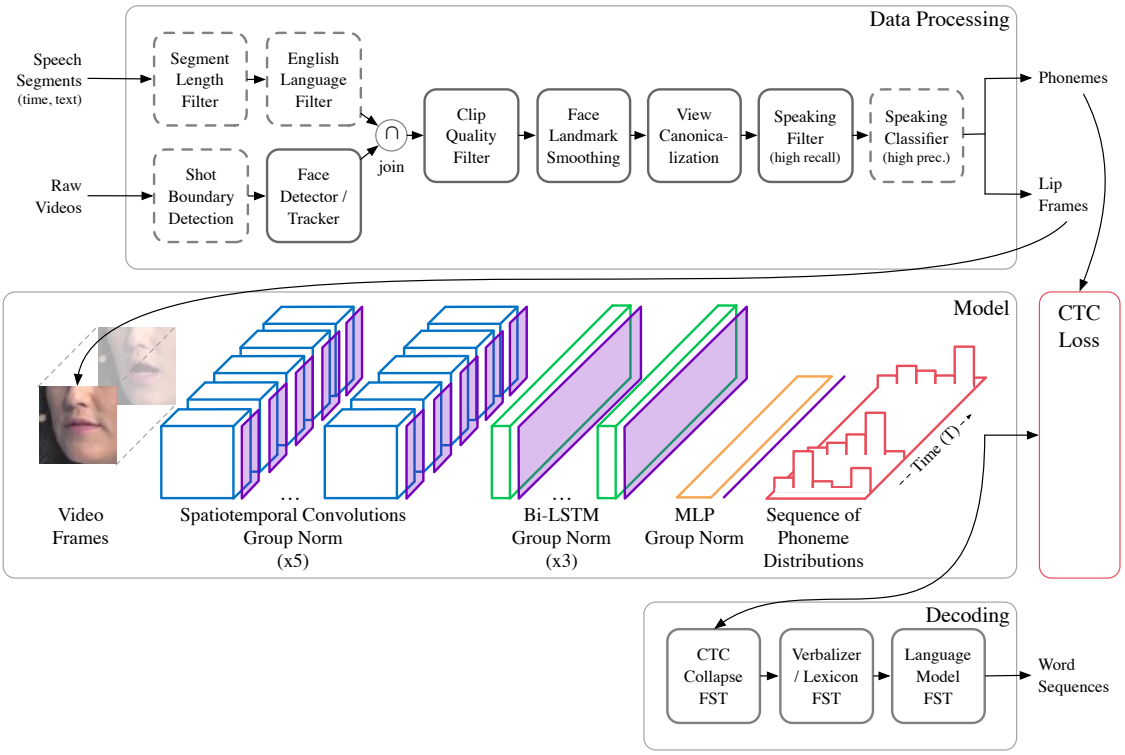


Figure 4.1: The full visual speech recognition system introduced by this work consists of a data processing pipeline that generates lip and phoneme clips from YouTube videos (see Section 4.4), and a scalable deep neural network for phoneme recognition combined with a phoneme-to-word decoding module used for inference (see Section 3.3).

phoneme prediction and word decoding as is often done in speech recognition, we are able to arbitrarily extend the vocabulary without retraining the neural network. Details of our model and this decoding process are given in Section 4.5.

Finally, this entire lipreading system results in an unprecedented WER of 40.9% as measured on a held-out set from our dataset. In comparison, professional lipreaders achieve either 86.4% or 92.9% WER on the same dataset, depending on the amount of context given. Similarly, previous state-of-the-art approaches such as variants of *LipNet* (Assael* et al., 2016) and of *Watch, Attend, and Spell* (WAS) (Chung et al., 2017) demonstrated WERs of only 89.8% and 76.8% respectively.

4.2 Potential medical applications

An important motivation for visual-only speech recognition is its potential to help those who lost their voice. As a consequence of injury or disease and its associated treatment, millions of people worldwide have communication problems preventing them from generating sound. As hearing aids and cochlear transplants have transformed the lives of people with hearing loss, there is potential for lip

reading technology to provide alternative communication strategies for people who have lost their voice.

Aphonia is the inability to produce voiced sound. It may result from injury, paralysis, removal or other disorders of the larynx. Common examples of primary aphonia include bilateral recurrent laryngeal nerve damage as a result of thyroidectomy *(removal of the thyroid gland and any tumour)* for thyroid cancer, laryngectomy *(surgical removal of the voice box)* for laryngeal cancers, or tracheostomy *(the creation of an alternate airway in the neck bypassing the voicebox)*.



Figure 4.2: V2P could be helpful for performing silent speech recognition for those with aphonia[1].

Dysphonia is difficulty in speaking due to a physical disorder of the mouth, tongue, throat, or vocal cords. Unlike aphonia, patients retain some ability to speak. For example, in Spasmodic dysphonia, a disorder in which the laryngeal muscles go into periods of spasm, patients experience breaks or interruptions in the voice, often every few sentences, which can make a person difficult to understand.

We see this work having potential medical applications for patients with aphonia or dysphonia in at least two distinct settings. Firstly, an acute care setting (i.e. a hospital with an emergency room and an intensive care unit), patients frequently undergo elective (planned) or emergency (unplanned) procedures (e.g. Tracheostomy) which may result in aphonia or dysphonia. In the U.S. 103,925 tracheostomies were performed in 2014, resulting in an average hospital stay of 29 days (HCUPnet, 2014). Similarly, in England and Wales 15,000 tracheostomies are performed each year (The Health Foundation, 2014).

Where these procedures are unplanned, there is often no time or opportunity to psychologically prepare the patient for their loss of voice, or to teach the patient alternative communication strategies. Some conditions that necessitate tracheotomy, such as high spinal cord injuries, also affect limb function, further hampering alternative communication methods such as writing.

Even where procedures are planned, such as for head and neck cancers, despite preparation of the patient through consultation with a speech and language therapist,

[1] `https://www.youtube.com/watch?v=FwOLHtHrVbc`

many patients find their loss of voice highly frustrating especially in the immediate post-operative period.

Secondly, where surgery has left these patients cancer-free, they may live for many years, even decades without the ability to speak effectively, in these patients we can envisage that they may use this technology in the community, after discharge from hospital. While some patients may either have tracheotomy reversed, or adapt to speaking via a voice prosthesis, electro-larynx or esophageal speech, many patients do not achieve functional spoken communication. Even in those who achieve good face-to-face spoken communication, few laryngectomy patients can communicate effectively on the telephone, and face the frequent frustration of being hung-up on by call centres and others who do not know them.

Acute care applications. It is widely acknowledged that patients with communication disabilities, including speech impairment or aphonia can pose significant challenges in the clinical environment, especially in acute care settings, leading to potentially poorer quality of care (Morris and Kho, 2014). While some patients will be aware prior to surgery that they may wake up unable to speak, for many patients in the acute setting (e.g. Cervical Spinal Cord Injury, sudden airway obstruction) who wake up following an unplanned tracheotomy, their sudden inability to communicate can be phenomenally distressing.

Community applications. Patients who are discharged from hospital without the ability to speak, or with poor speech quality, face a multitude of challenges in day-to-day life which limits their independence, social functioning and ability to seek employment.

We hypothesize that the application of technology capable of lip-reading individuals with the ability to move their facial muscles, but without the ability to speak audibly could significantly improve quality of life for these patients. Where the application of this technology improves the person's ability to communicate over the telephone, it would enhance not only their social interactions, but also their ability to work effectively in jobs that require speaking over the phone.

Finally, in patients who are neither able to speak, nor to move their arms, this technology could represent a step-change in terms of the speed at which they can communicate, as compared to eye-tracking or facial muscle based approaches in use today.

4.3 Related work

While there is a large body of literature on automated lipreading, much of the early work focused on single-word classification and relied on substantial prior knowledge (Chu and Huang, 2000; Matthews et al., 2002; Pitsikalis et al., 2006; Lucey and Sridharan, 2006; Papandreou et al., 2007; Zhao et al., 2009; Gurban and Thiran, 2009; Papandreou et al., 2009). For example, Goldschen et al. (1997) predicted continuous sequences of tri-visemes using a traditional HMM model with visual features extracted from a codebook of clustered mouth region images. The predicted visemes were used to distinguish sentences from a set of 150 possible sentences. Furthermore, Potamianos et al. (1997) predict words and sequences digits using HMMs, Potamianos and Graf (1998) introduce multi-stream HMMs, and Potamianos et al. (1998) improve the performance by using visual features in addition to the lip contours. Later, Chu and Huang (2000) used coupled HMMs to jointly model audio and visual streams to predict sequences of digits. Neti et al. (2000) used HMMs for sentence-level speech recognition in noisy environments of the IBM ViaVoice dataset by fusing handcrafted visual and audio features. More recent attempts using traditional speech, vision and machine learning pipelines include the works of Gergen et al. (2016), Paleček (2017), and Hassanat (2011) and Bear and Harvey (2016). For further details, we refer the reader to the survey material of Potamianos et al. (2004) and Zhou et al. (2014).

However, as noted by Zhou et al. (2014) and Assael* et al. (2016), until recently generalization across speakers and extraction of motion features have been considered open problems. Advances in deep learning have made it possible to overcome these limitations, but most works still focus on single-word classification, either by learning visual-only representations (Hinton et al., 2012; Chung and Zisserman, 2016b; Wand et al., 2016; Stafylakis and Tzimiropoulos, 2017; Wand and Schmidhuber, 2017), multimodal audio-visual representations (Ngiam et al., 2011; Sui et al., 2015; Ninomiya et al., 2015; Petridis and Pantic, 2016; Petridis et al., 2017), or combining deep networks with traditional speech techniques (e.g. HMMs and GMM-HMMs) (Noda et al., 2014; Koller et al., 2015; Almajai et al., 2016; Takashima et al., 2016).

LipNet (Chapter 3) was the first end-to-end model to tackle sentence-level lipreading by predicting character sequences. The model combined spatiotemporal convolutions with gated recurrent units (GRUs) and was trained using the CTC loss function. LipNet was evaluated on the GRID corpus (Cooke et al., 2006), a limited grammar and vocabulary dataset consisting of 28 hours of 5-word sentences, where it achieved 4.8% and 11.4% WER in overlapping and unseen speaker evaluations

respectively. By comparison, the performance of competent human lipreaders on GRID was 47.7%. LipNet is the closest model to our neural network. Several similar architectures were subsequently introduced in the works of Thanda and Venkatesan (2017) who study audio-visual feature fusion, Koumparoulis et al. (2017) who work on a small subset of 18 phonemes and 11 words to predict digit sequences, and Xu et al. (2018) who presented a model cascading CTC with attention.

Chung et al. (2017) were the first to use sequence-to-sequence models with attention to tackle audio-visual speech recognition with a real-world dataset. The model "Watch, Listen, Attend and Spell" (WLAS), consists of a visual (WAS) and an audio (LAS) module. To evaluate WLAS, the authors created LRS, the largest dataset at that point with approximately 246 hours of clips from BBC news broadcasts, and introduced an efficient video processing pipeline. The authors reported 50.2% WER, with the performance of professional lipreaders being 87.6% WER. Chung and Zisserman (2017) extended the work to multi-view sentence-level lipreading, achieving 62.8% WER for profile views and 56.4% WER for frontal views. Both Chung et al. (2017) and Chung and Zisserman (2017) pre-learn features with the audio-video synchronization classifier of Chung and Zisserman (2016a), and fix these features in order to compensate for the large memory requirements of their attention networks. Contemporaneously with our work, Afouras et al. (2018b) presented LRS3-TED, a dataset generated from English language talks available online. Using pre-learned features Afouras et al. (2018a) presented a seq2seq and a CTC architecture based on character-level self-attention transformer models. On LRS3-TED, these models achieved a WER of 57.9% and 61.8% respectively. Other related advances include works using vision for silent speech reconstruction (Le Cornu and Milner, 2017; Ephrat and Peleg, 2017; Akbari et al., 2017; Gabbay et al., 2017) and for separating an audio signal to individual speech sources (Ephrat et al., 2018; Afouras et al., 2018c).

In contrast to the approach of Assael* et al. (2016), our model (V2P) uses a network to predict a sequence of phoneme distributions which are then fed into a decoder to produce a sequence of words. This flexible design enables us to easily accommodate very large vocabularies, and in fact we can extend the size of the vocabulary without having to retrain the deep network. Unlike previous work, V2P is memory and computationally efficient without requiring pre-trained features (Chung et al., 2017; Chung and Zisserman, 2017).

Finally, the data processing pipeline used in this work results in a significantly larger and more diverse training dataset than in all previous efforts. While the first large-vocabulary lipreading dataset was IBM ViaVoice (Neti et al., 2000), more

Dataset	Utter.	Hours	Vocab
GRID	33,000	28	51
IBM ViaVoice	17,111	35	10,400
MV-LRS	74,564	∼155	14,960
LRS	118,116	∼246	17,428
LRS3-TED	∼165,000	∼475	∼57,000
LSVSR	**2,934,899**	**3,886**	**127,055**

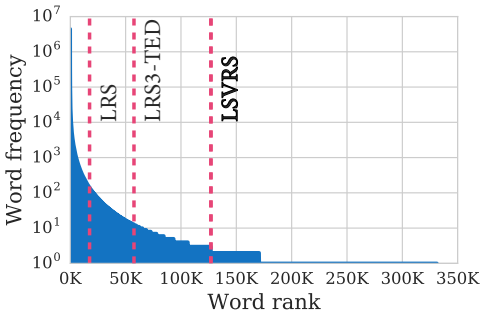


Figure 4.3: Left: A comparison of sentence-level (word sequence) visual speech recognition datasets. Right: Frequency of words in the LSVSR dataset in decreasing order of occurrence; approximately 350K words occur at least 3 times. We used this histogram to select a vocabulary of 127,055 words as it captures most of the mass.

recently the far larger LRS and MV-LRS datasets (Chung et al., 2017; Chung and Zisserman, 2017) were generated from BBC news broadcasts, and the LRS3-TED dataset was generated from conference talks. MV-LRS and LRS3-TED are the only publicly available large-vocabulary datasets, although both are limited to academic usage. In comparison, our dataset (LSVSR) is an order of magnitude greater than any previous dataset with 3,886 hours of audio-video-text pairs. In addition, the content is much more varied (i.e. not news-specific), resulting in a 2.2× larger vocabulary of 127,055 words. Figure 4.3 shows a comparison of sentence-level (word sequence) visual speech recognition datasets.

4.4 A data pipeline for large-scale visual speech recognition

In this section we discuss the data processing pipeline, again illustrated in Figure 4.1, used to create the LSVSR dataset. Our pipeline makes heavy use of large-scale parallel processing and is implemented as a number of independent modules and filters on top of FlumeJava (Chambers et al., 2010). In particular, our dataset is extracted from public YouTube videos. This is a common strategy for building datasets in ASR and speech enhancement (Liao et al., 2013; Liao et al., 2015; Kuznetsov et al., 2016; Soltau et al., 2017; Ephrat et al., 2018).

In our case, we build on the work of Liao et al. (2013) to extract audio clips paired with transcripts, yielding 140,000 hours of audio segments. After post-processing we obtain a dataset consisting of paired video and phoneme sequences, where video sequences are represented as identically-sized frames (here, 128×128)

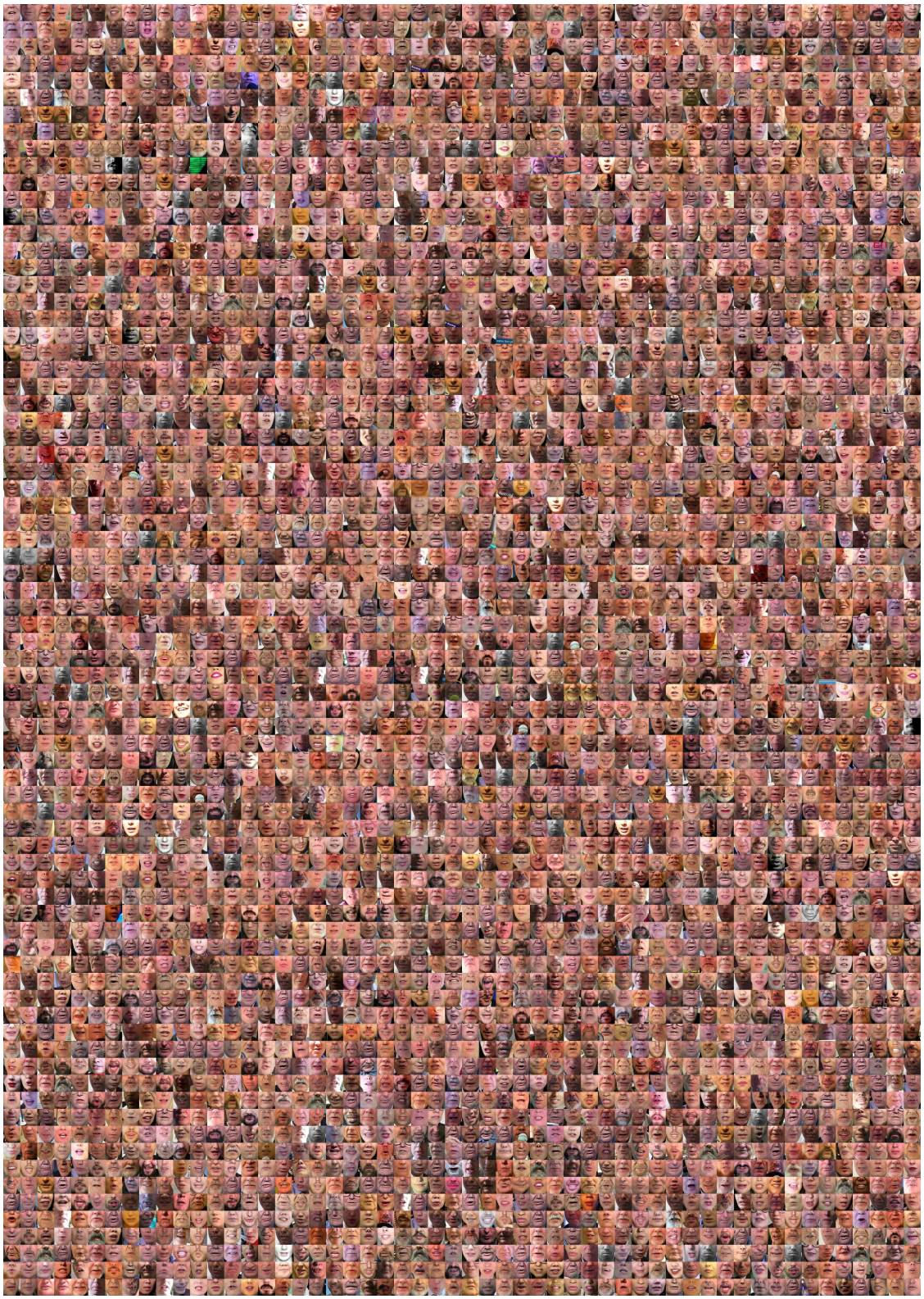


Figure 4.4: Random sample of test-set lip images from LSVSR. This illustrates the substantial diversity in our dataset.

stacked in the time-dimension. Although our pipeline is used to process clips pre-selected from YouTube (Liao et al., 2013), only about 2% of clips satisfy our filtering criteria. Finally, by eliminating the components marked by dashes in Figure 4.1, i.e. those components whose primary use are in producing paired training data, this same pipeline can be used in combination with a trained model to predict word sequences from raw videos. In what follows we describe the individual components that make up this pipeline.

Length filter, language filter. The duration of each segment extracted from YouTube is limited to between 1 and 12 seconds, and the transcripts are filtered through a language classifier (Salcianu et al., 2018) to remove non-English utterances. For evaluation, we further remove the utterances containing fewer than 6 words. Finally, the aligned phoneme sequences are obtained via a standard forced alignment approach using a lexicon with multiple pronunciations (Liao et al., 2013). The phonetic alphabet is a reduced version of X-SAMPA (Wells, 1995) with 40 phonemes plus silence.

Raw videos, shot boundary detection, face detection. Constant spatial padding in each video segment is eliminated before a standard, thresholding color histogram classifier (Mas and Fernandez, 2003) identifies and removes segments containing shot boundaries. FaceNet (Schroff et al., 2015) is used to detect and track faces in every remaining segment.

Clip quality filter. Speech segments are joined with the set of tracked faces identified in the previous step and filtered based on the quality of the video, removing blurry clips and clips including faces with an eye-to-eye width of less than 80 pixels. Frame rates lower than 23fps are also eliminated (Saitoh and Konishi, 2010; Taylor et al., 2014). We allow a range of input frame rates—varying frame rates has an effect similar to different speaking paces—however, frame rates above 30fps are downsampled.

Face landmark smoothing. The segments are processed by a face landmark tracker and the resulting landmark positions are smoothed using a temporal Gaussian kernel. Intuitively, this simplifies learning filters for the 3D convolution layers by reducing spatiotemporal noise. Empirically, our preliminary studies showed smoothing was crucial for achieving optimal performance. Next, following previous literature (Chung et al., 2017), we keep segments where the face yaw and pitch remain within $\pm 30°$. Models trained outside this range perform worse (Chung and Zisserman, 2017).

View canonicalization. We obtain canonical faces using a reference canonical face model and by applying an affine transformation on the landmarks. Then,

we use a thumbnail extractor which is configured to crop the area around the lips of the canonical face.

Speaking filter. Using the extracted and smoothed landmarks, minor lip movements and non-speaking faces are discarded using a threshold filter. This process involves computing the mouth openness in all frames, normalizing by the size of the face bounding box, and then thresholding on the standard deviation of the normalized openness. This classifier has very low computational cost, but high recall, e.g. voice-overs are not handled. Its contribution was very important to process approximately 16 years of audio-video-text pairs within reasonable time.

Speaking classifier. As a final step, we build *V2P-Sync*, a neural network architecture to verify the audio and video channel alignment inspired by the work of Chung and Zisserman (2016a) and Torfi et al. (2017). However, V2P-Sync takes advantage of face landmark smoothing, 3D convolutions, and high resolution inputs. V2P-Sync uses longer time segments as inputs and spatiotemporal convolutions as compared to the spatial-only convolutions of Chung and Zisserman, and view canonicalization and higher resolution inputs (128×128 vs 100×60) as compared to Torfi et al. These characteristics facilitate the extraction of temporal features which is key to our task. V2P-Sync, takes as input a pair of a log mel-spectrogram and 9 grayscale video frames and produces an embedding for each using two separate neural network architectures. If the Euclidean distance of the audio and video embeddings is less than a given threshold the pair is classified as synchronized. The architecture is trained using a contrastive loss similar to Chung and Zisserman. Since there is no labeled data for training, the initial unfiltered pairs are used as positive samples with negative samples generated by randomly shifting the video of an unfiltered pair. After convergence the dataset is filtered using the trained model, which is then fine-tuned on the resulting subset of the initial dataset. The final model is used to filter the dataset a second time, achieving an accuracy of 81.2%. This accuracy is improved as our audio-video pairs are processed by sliding V2P-Sync on 100 equally spaced segments and their scores are averaged.

The overall architecture of V2P-Sync is shown in Figure 4.5. The architecture of the video and audio subnetworks are shown in Section 4.4 and Table 4.2. Their parameters are optimized using a batch size of 128, batch normalization, and Adam (Kingma and Ba, 2014) with a learning rate of 10^{-4} and default hyperparameters: first and second momentum coefficients 0.9 and 0.999 respectively, and $\epsilon = 10^{-8}$ for numerical stability.

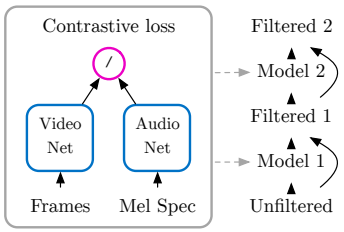


Figure 4.5: V2P-Sync takes as input an audio-visual pair and produces embeddings for both, then using the Euclidean distance of the embeddings the pair is classified as in-sync or not. The model (Model 1) is a) trained on the initial dataset (Unfiltered), then b) it is used to filter the dataset (Filtered 1), c) fine-tuned on the filtered dataset (Model 2), and d) used to filter the dataset for the second time (Filtered 2).

Layer	Filter size	Stride	Output channels	Input
conv1	$3 \times 3 \times 3$	$1 \times 2 \times 2$	16	$9 \times 128 \times 128 \times 1$
pool1	$1 \times 2 \times 2$	$1 \times 2 \times 2$		$7 \times 63 \times 63 \times 16$
conv2	$3 \times 3 \times 3$	$1 \times 1 \times 1$	32	$7 \times 31 \times 31 \times 16$
pool2	$1 \times 2 \times 2$	$1 \times 2 \times 2$		$5 \times 29 \times 29 \times 32$
conv3	$3 \times 3 \times 3$	$1 \times 1 \times 1$	64	$5 \times 14 \times 14 \times 32$
pool3	$1 \times 2 \times 2$	$1 \times 2 \times 2$		$3 \times 12 \times 12 \times 64$
conv4	$3 \times 3 \times 3$	$1 \times 1 \times 1$	128	$3 \times 6 \times 6 \times 64$
pool4	$1 \times 2 \times 2$	$1 \times 2 \times 2$		$1 \times 4 \times 4 \times 128$
fc5		$1 \times 1 \times 1$	256	512
fc6		$1 \times 1 \times 1$	64	256

Table 4.1: V2P-Sync video embedding neural network architecture.

Table 4.2: V2P-Sync audio embedding neural network architecture.

Layer	Support	Stride	Filters	Input
conv1	3×5	1×1	16	$16 \times 40 \times 1$
pool1	1×2	1×2	$14 \times 36 \times 16$	
conv2	3×4	1×1	32	$14 \times 36 \times 16$
conv3	3×4	1×1	32	$12 \times 15 \times 32$
pool3	1×2	1×2	$10 \times 12 \times 32$	
conv4	3×3	1×1	64	$10 \times 6 \times 32$
conv5	3×3	1×1	64	$8 \times 4 \times 64$
conv6	3×2	1×1	128	$6 \times 2 \times 64$
fc7		1×1	256	512
fc8		1×1	64	256

4.5 An efficient spatiotemporal model of visual speech recognition

This work introduces the V2P model, which consists first of a *3d convolutional module* for extracting spatiotemporal features from a given video clip. These features are then aggregated over time with a *temporal module* which outputs a sequence of phoneme distributions. Given input video clips and target phoneme sequences the model is trained using the *CTC* loss function. Finally, at test-time, a *decoder* based on finite state transducers (FSTs) is used to produce a word sequence given a sequence of phoneme distributions.

Neural network architecture. Although the use of optical-flow filters as inputs is commonplace in lipreading (Mase and Pentland, 1991; Gray et al., 1997; Yoshinaga et al., 2003; Tamura et al., 2004; Wang et al., 2008; Shaikh et al., 2010), in this work we designed a vision module based on VGG (Simonyan and Zisserman, 2015) to explicitly address motion feature extraction. We adapted VGG to make it volumetric, which proved crucial in our preliminary empirical evaluation and has been established in previous literature (Assael* et al., 2016). The intuition behind this is the importance of spatiotemporal relationships in human visual speech recognition, e.g. measuring how lip shape changes over time. Furthermore, the receptive field of the vision module is 11 video frames, roughly 0.36–0.44 seconds, or around twice the typical duration of a phoneme.

One of the main challenges in training a large vision module is finding an effective balance between performance and the imposed constraints of GPU memory. Our vision module consists of 5 convolutional layers with $[64, 128, 256, 512, 512]$ filters. By profiling a number of alternative architectures, we found that high memory usage typically came from the first two convolutional layers. To reduce the memory footprint we limit the number of convolutional filters in these layers, and since the frame is centered around the lips, we omit spatial padding. Since phoneme sequences can be quite long, but with relatively low frame rate (approximately 25–30 fps), we maintain padding in the temporal dimension and always convolve with unit stride in order to avoid limiting the number of output tokens. Despite tuning the model to reduce the number of activations, we are still only able to fit 2 batch elements on a GPU. Hence, we distribute training across 64 workers in order to achieve a batch size of 128. Due to communication costs, batch normalization is expensive if one wants to aggregate the statistics across all workers, and using only two examples per batch results in noisy normalization statistics. Thus, instead of batch normalization, we use group normalization (Wu and He, 2018), which

divides the channels into groups and computes the statistics within these groups. This provides more stable learning regardless of batch size.

The outputs of the convolutional stack are then fed into a temporal module which performs longer-scale aggregation of the extracted features over time. In constructing this component we evaluated a number of recurrent neural network and dilated convolutional architectures, the latter of which are evaluated later as baselines. The best architecture presented performs temporal aggregation using a stack of 3 bidirectional LSTMs (Hochreiter and Schmidhuber, 1997) with a hidden state of 768, interleaved with group normalization. The output of these LSTM layers is then fed through a final MLP layer to produce a sequence of exactly T conditionally independent phoneme distributions $p(u_t|\mathbf{x})$. This entire model is then trained using the CTC loss we describe next.

This model architecture is similar to that of the closest related work, LipNet (Chapter 3), but differs in a number of crucial ways. In comparison to our work, LipNet used GRU units and dropout, both of which we found to perform poorly in preliminary experiments. Our model is also much bigger: LipNet consists of only 3 convolutional layers of $[32, 64, 96]$ filters and 3 GRU layers with hidden state of size 256. Although the small size of LipNet means that it does not require any distributed computation to reach effective batch sizes, we will see that this drop in size coincides with a similar drop in performance. Finally, while both models use a CTC loss for training, the architecture used in V2P is trained to predict phonemes rather than characters; as we argue shortly this provides V2P with a much simpler mechanism for representing word uncertainty.

The network's architecture and hyperparameters are specified in Section 4.5. Its parameters are optimized using Adam (Kingma and Ba, 2014) with a learning rate of 10^{-4} and default hyperparameters: first and second momentum coefficients 0.9 and 0.999 respectively, and $\epsilon = 10^{-8}$ for numerical stability. Furthermore, to accelerate learning, a curriculum schedule limits the video duration, starting from 2 seconds and gradually increasing to a maximum length of 12 seconds over 200,000 training steps. Finally, image transformations are also applied to augment the image frames to help improve invariance to filming conditions. This is accomplished by first randomly mirroring the videos horizontally, followed by random changes to brightness, contrast, saturation, and hue.

Connectionist temporal classification (CTC). CTC is a loss function for the parameterization of distributions over sequences of label tokens, without requiring alignments of the input sequence to the label tokens (Graves et al., 2006). For more details, see Section 1.5.

Layer	Filter size	Stride	Output channels	Input
conv1	$3 \times 3 \times 3$	$1 \times 2 \times 2$	64	$\mathrm{T} \times 128 \times 128 \times 3$
pool1	$1 \times 2 \times 2$	$1 \times 2 \times 2$		$\mathrm{T} \times 63 \times 63 \times 64$
conv2	$3 \times 3 \times 3$	$1 \times 1 \times 1$	128	$\mathrm{T} \times 31 \times 31 \times 64$
pool2	$1 \times 2 \times 2$	$1 \times 2 \times 2$		$\mathrm{T} \times 29 \times 29 \times 128$
conv3	$3 \times 3 \times 3$	$1 \times 1 \times 1$	256	$\mathrm{T} \times 14 \times 14 \times 128$
pool3	$1 \times 2 \times 2$	$1 \times 2 \times 2$		$\mathrm{T} \times 12 \times 12 \times 256$
conv4	$3 \times 3 \times 3$	$1 \times 1 \times 1$	512	$\mathrm{T} \times 6 \times 6 \times 256$
conv5	$3 \times 3 \times 3$	$1 \times 1 \times 1$	512	$\mathrm{T} \times 4 \times 4 \times 512$
pool5	$1 \times 2 \times 2$	$1 \times 1 \times 1$		$\mathrm{T} \times 2 \times 2 \times 512$
bilstm6			768×2	$\mathrm{T} \times 512$
bilstm7			768×2	$\mathrm{T} \times 1536$
bilstm8			768×2	$\mathrm{T} \times 1536$
fc9			768	$\mathrm{T} \times 1536$
fc10			$41 + 1$	$\mathrm{T} \times 768$

Table 4.3: V2P architecture details.

Since CTC prevents us from using autoregressive connections to handle inter-timestep dependencies of the label sequence, the marginal distributions produced at each timestep of the temporal module are conditionally independent, as pointed out above. Therefore, to restore temporal dependency of the labels at test-time, CTC models are typically decoded with a beam search procedure that combines the probabilities with that of a language model.

Rationale for phonemes and CTC. In speech recognition, whether on audio or visual signals, there are two main sources of uncertainty: uncertainty in the sounds that are in the input, and uncertainty in the words that correspond to these sounds. Recall that our goal is to map from the input to a sequence of words. This suggests modelling

$$\begin{aligned} p(\text{words}|\mathbf{x}) &= \sum_{\text{phoneme sequence } p} p(\text{words}|\text{phonemes} = p)p(\text{phonemes} = p|\mathbf{x}) \\ &\approx p(\text{words}|\text{phonemes})p(\text{phonemes}|\mathbf{x}), \end{aligned}$$

where the approximation is by the assumption that a given word sequence often has a single or dominant pronunciation. While previous work uses CTC to model characters given audio or visual input directly (Assael* et al., 2016; Amodei et al., 2016), we argue this is problematic as the conditional independence of CTC timesteps

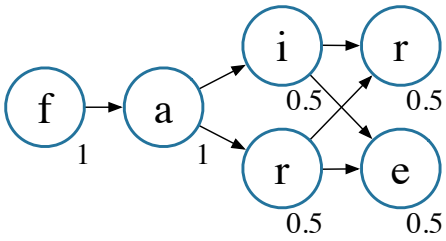


Figure 4.6: Example homophone issue when modelling characters with CTC.

means that the temporal module must assign a high probability to a single sequence in order to not produce spurious modes in the CTC distribution.

To explain why modeling characters with CTC is problematic, consider two character sequences "fare" and "fair" that are homophones, i.e. they have the same pronunciation (i.e. /fɛ:/). The difficulty we will describe is independent of the model used, so we will consider a simple unconditional model where each character c is assigned probability given by the parameters $\pi_t^c = P(u_t = c)$ and the probability of a sequence is given by its product, e.g. $p(\text{fare}) = \pi_1^{\text{f}}\pi_2^{\text{a}}\pi_3^{\text{r}}\pi_4^{\text{e}}$. The maximum likelihood estimate, $\arg\max_\pi p(\text{fare})p(\text{fair})$, however, assigns equal 1/4 probability to each of "fare", "fair", "faie", "farr", as shown in Figure 4.6, resulting in two undesirable words.

Ultimately this difficulty arises due to the independence assumption of CTC and the many-to-many mapping of characters to words.[2] This same difficulty arises if we replace the parameters above with the outputs of a network mapping from videos to tokens.

An additional source of uncertainty in visual speech recognition is introduced by the fact that the information required to disambiguate different phonemes is not visible (e.g. the position of the tongue). The resulting visually similar phonemes are called visemes. This can be seen in Section 4.6.3 where the ratio of insertions and substitutions when computing the edit distance of a visual model is substantially higher than the ratio of an audio model trained on the same dataset. Furthermore, recent literature suggests that phoneme-to-viseme mappings may differ per speaker (Bear and Harvey, 2017), making it difficult to incorporate this knowledge in the design. Thus, using phonemes, which have a one-to-many mapping to words, allows the temporal model to only model visual uncertainty, and the word uncertainty can instead be handled by the decoder described below.

Alternatively to using phonemes with CTC, some previous work solves this problem using RNN transducers (Rao et al., 2017) or sequence-to-sequence with

[2]Languages such as Korean, where there is a nearly one-to-one correspondence between pronunciation and orthography, do not give rise to such discrepancies.

attention (Chung et al., 2017), which jointly model all sources of uncertainty. However, Prabhavalkar et al. (2017) showed in the context of acoustic speech recognition that these models were unable to significantly outperform a baseline CTC model (albeit using context-dependent phonemes and further sequence-discriminative training) when combined with a decoding pipeline similar to ours. Hence, for reasons of performance and easier model training, especially important with our large model, we choose to output phonemes rather than words or characters directly. Additionally, and crucial for many applications, CTC also provides extra flexibility over alternatives. The fact that the lexicon (phoneme to word mapping) and language model are separate and part of the decoder, affords one the ability to trivially change the vocabulary and language model (LM) arbitrarily. This allows for visual speech recognition in narrower domains or updating the vocabulary and LM with new words without requiring retraining of the phoneme recognition model. This is nontrivial in other models, where the language model is part of the RNN.

Decoding. As described earlier, our model produces a sequence of phoneme distributions; given these distributions we use an industry-standard decoding method using finite state transducers (FSTs) to arrive at word sequences. Such techniques are extensively used in speech recognition (e.g. Miao et al., 2015; McGraw et al., 2016); we refer the reader to the thorough presentation of Mohri et al. (2002). In our work we make use of a combination of three individual (weighted) FSTs, or WFSTs. The first *CTC postprocessing FST* removes duplicate symbols and CTC blanks. Next, a *lexicon* FST maps input phonemes to output words. Third, an *n-gram language model* with backoff can be represented as a WFST from words to words. In our case, we use a 5-gram model with Katz backoff with about 50 million n-grams and a vocabulary size of about one million. The composition of these three FSTs results another WFST transducing from phoneme sequences to (reweighted) word sequences. Finally, a search procedure is employed to find likely word sequences from phoneme distributions.

4.6 Evaluation

We examine the performance of V2P trained on LSVSR with hyperparameters tuned on a validation set. We evaluate it on a held-out test set roughly 37 minutes long, containing approximately 63,000 video frames and 7100 words. We also describe and compare against a number of alternate methods from previous work. In particular, we show that our system gives significant performance improvements over professional lipreaders as well previous state-of-the-art methods for visual

speech recognition. Except for V2P-NoLM, all models used the same 5-gram word-level language model during decoding. To construct the validation and test sets we removed low-quality blurry videos by thresholding the variance of the Laplacian of each frame (Pech-Pacheco et al., 2000); we kept them in the training set as a form of data augmentation.

Professional lipreaders. We consulted a professional lipreading company to measure the difficulty of LSVSR and hence the impact that such a model could have. Since the inherent ambiguity in lipreading necessitates relying on context, we conducted experiments both with and without context. In both cases we generate modified clips from our test set, but cropping the whole head in the video, as opposed to just the mouth region used by our model. The lipreaders could view the video up to 10 times, at half or normal speed each time. To measure without-context performance, we selected clips with transcripts that had at least 6 words. To measure how much context helps performance, we selected clips with at least 12 words, and presented to the lipreader the first 6 words, the title, and the category of the video, then asked them to transcribe the rest of the clip. The lipreaders transcribed a subset of our test set containing 153 and 274 videos with and without context, respectively.

Audio-Ph. For an approximate bound on performance, we train an audio speech recognition model on the audio of the utterances. The architecture is based on Deep Speech 2 (Amodei et al., 2016), but trained to predict phonemes rather than characters.

Baseline-LipNet-Ch. Using our training setup, we replicate the character-level CTC architecture of LipNet (Assael* et al., 2016). As with the phoneme models, we use an FST decoding pipeline and the same language model, but instead of a phoneme-based lexicon we use a character-level one as described in Miao et al. (2015).

Baseline-LipNet-Ph. We also train LipNet to predict phonemes, still with CTC and using the same FST-based decoding pipeline and language model.

Baseline-LipNet-Large-Ph. Recall from the earlier discussion that LipNet uses dropout, whereas V2P makes heavy use of group normalization, crucial for our small batches per worker. For a fair size-wise comparison, we introduce a replica of V2P, that uses GRUs, dropout, and no normalization.

Baseline-Seq2seq-Ch. Using our training setup, we compared to a variant of the previous state-of-the-art sequence-to-sequence architecture of WAS that predicts character sequences (Chung et al., 2017). Although their implementation was followed as closely as possible, training end-to-end quickly exceeded the memory limitations of modern GPUs. To work around these problems, the authors kept the convolutional weights fixed using a pretrained network from audio-visual

Table 4.4: Performance evaluation on LSVSR test set. Columns show phoneme, character, and word error rates, respectively. Standard deviations are bootstrap estimates.

Method	Params	PER	CER	WER
Professional w/o context	–	–	–	92.9 ± 0.9
Professional w/ context	–	–	–	86.4 ± 1.4
Audio-Ph	58M	12.5 ± 0.5	11.5 ± 0.6	18.3 ± 0.9
Baseline-LipNet-Ch	7M	–	64.6 ± 0.5	93.0 ± 0.6
Baseline-LipNet-Ph	7M	65.8 ± 0.4	72.8 ± 0.5	89.8 ± 0.5
Baseline-Seq2seq-Ch	15M	–	49.9 ± 0.6	76.8 ± 0.8
Baseline-LipNet-Large-Ph	40M	53.0 ± 0.5	54.0 ± 0.8	72.7 ± 1.0
V2P-FullyConv	29M	41.3 ± 0.6	36.7 ± 0.9	51.6 ± 1.2
V2P-NoLM	49M	33.6 ± 0.6	34.6 ± 0.8	53.6 ± 1.0
V2P	**49M**	$\mathbf{33.6 \pm 0.6}$	$\mathbf{28.3 \pm 0.9}$	$\mathbf{40.9 \pm 1.2}$

synchronization classification (Chung and Zisserman, 2016a), which we were unable to use as their network inputs were processed differently. Instead, we replace the 2D convolutional network with the *improved* lightweight 3D visual processing network of V2P. From our empirical evaluation, including preliminary experiments not reported here and as shown by earlier work (Assael* et al., 2016), we believe that the 3D spatiotemporal aggregation of features benefits performance. After standard beam search decoding, we use the same 5-gram word LM as used for the CTC models to perform reranking.

V2P-FullyConv. Identical to V2P, except the LSTMs in the temporal aggregation module are replaced with 6 dilated temporal convolution layers with a kernel size of 3 and dilation rates of [1,1,2,4,8,16], yielding a fully convolutional model with 12 layers.

V2P-NoLM. Identical to V2P, except during decoding, where the LM is replaced with a dictionary consisting of 100k words. The words are then weighted by their smoothed frequency in the training data, essentially a uni-gram language model.

4.6.1 Results

Table 4.4 shows the phoneme error rate, character error rate, and word error rate for all of the models, and the number of parameters for each. The error rates are computed as the sum of the edit distances of the predicted and ground-truth sequence pairs divided by total ground-truth length. We also compute and display the standard error associated with each rate, estimated by bootstrap sampling.

These results show that the variant of LipNet tested in this work is approximately able to perform on-par with professional lipreaders with WER of 86.4% and 89.8%

respectively, even when the given professional is given additional context. Similarly, we see that the WAS variant provides a substantial reduction to this error, resulting in a WER of 76.8%. However, the full V2P method presented in this work is able to further halve the WER, obtaining a value of 40.9% at testing time. Interestingly, we see that although the bi-directional LSTM provides the best performance, using a fully-convolutional network still results in performance that is significantly better than all previous methods. Finally, although we see that the full V2P model performs best, removing the language model results only in a drop of approximately 13 WER to 53.6%.

By predicting phonemes directly, we also side-step the need to design phoneme-to-viseme mappings (Bear and Harvey, 2017). The inherent uncertainty is instead modelled directly in the predictive distribution. For instance, using edit distance alignments of the predictions to the ground-truths, we can determine which phonemes were most frequently erroneously included or missed, as shown in Figure 4.8. Here we normalize the rates of deletions vs insertions, however empirically we saw that deletions were much more common than inclusions. Among these errors the most common include phonemes that are often occluded by the teeth (/d/, /n/, and /t/) as well as the most common English vowel /@/. Finally, by differentiating the likelihood of the phoneme sequence with respect to the inputs using guided backpropagation (Springenberg et al., 2014), we compute the saliency maps shown in the top row of Figure 4.10 as a white overlay. The entropy at each timestep of the phoneme predictive distribution is shown as well.

4.6.2 Phoneme confusions

To compute the confusion matrix and the insertion/deletion chart shown in the main text in Figure 4.8, we first compute the edit distance dynamic programming matrix between each predicted sequence of phonemes and the corresponding ground-truth. Then, a backtrace through this matrix gives an alignment of the two sequences, consisting of edit operations paired with positions in the prediction/ground-truth sequences.

Counting the correct phonemes and the substitutions yields the confusion matrix Figure 4.7. The reader can note that the diagonal is strongly dominant. A few groups are commonly confused as expected due to their visual similarity, such as {/d/, /n/, /t/}, and to a lesser extent {/b/, /p/}.

Counting insertions/deletions yields Figure 4.8 in the main text, showing which phonemes are most commonly omitted (deleted), or less frequently, erroneously inserted.

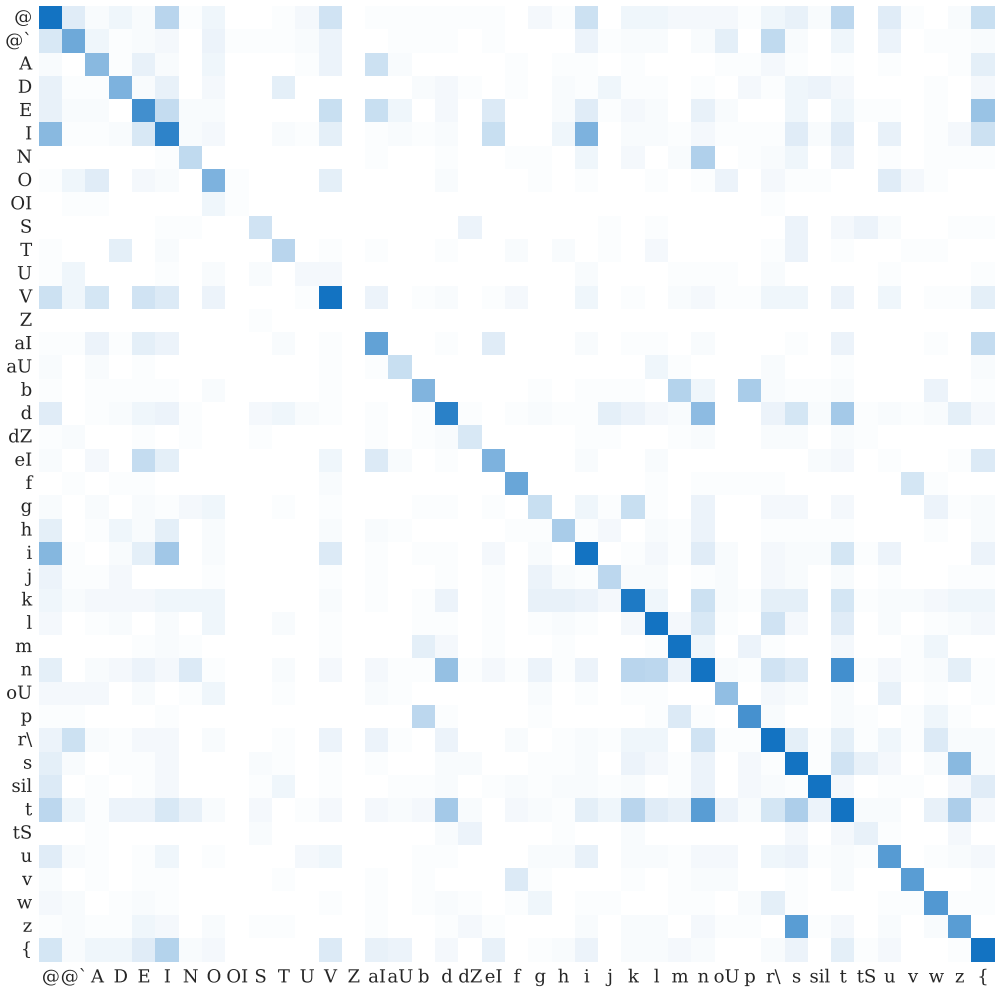


Figure 4.7: Phoneme confusion matrix for V2P, estimated by computing the edit distance alignment between each predicted sequence of phonemes and the corresponding ground-truth, and counting the correct phonemes and the substitutions. The diagonal values are scaled downwards to de-emphasize the correct phonemes. Blue indicates more substitutions occurred.



Figure 4.8: This heatmap shows which insertion and deletion errors were most common on the test set. Blue indicates more insertions or deletions occurred.

Table 4.5: Audio-ph and V2P phoneme deletions, insertions, and substitutions.

	Audio-ph		V2P	
Deletions	849	(38.9%)	1366	(10.2%)
Insertions	1245	(26.5%)	5378	(40.0%)
Substitutions	1109	(34.6%)	6697	(49.8%)
Total	3203		13441	

4.6.3 Phoneme deletions, insertions, substitutions

Using the same method for computing the edit distance as described in Section 4.6.2, Table 4.5 shows the absolute numbers of insertions/deletions/substitutions for the Audio-Ph and the V2P model. As it can be seen, the percentage of insertions and substitutions is substantially higher when using the visual channel (V2P) compared to the audio channel (Audio-Ph). Different phonemes can be visually identical and the information to disambiguate them is missing from the visual channel, this may result to a higher insertion and substitution rate.

4.6.4 Face rotation vs. performance heatmap

The following heatmap shows how performance of V2P varies over utterances with different amounts of head rotation, measured on the test set of LSVSR. An utterance is defined to be in a specific head rotation range defined by a minimum and maximum tilt and pan angle if the entire utterance lies fully in the specified angle ranges. The heatmap is then constructed by partitioning the space of tilt and pan angles into a grid, and measuring test set performance on the subset within the angle range of each grid cell.

As shown in Figure 4.9, it performs similarly at all pan and tilt angles in $[-30°, 30°]$, the range at which it was trained.

4.6.5 Saliency maps

In this visualization, we differentiate the likelihood of the phoneme sequence with respect to the model's inputs using guided backpropagation (Springenberg et al., 2014). This gives an image-shaped gradient at each timestep, which we can visualize as an overlay over the frames. This is shown as a white overlay. The entropy of the phoneme predictive distribution at each timestep is shown as well to illustrate confidence of the timestep-wise classifications, alongside the top 3 predictions at each frame. The CTC blank character is represented by '␣'.

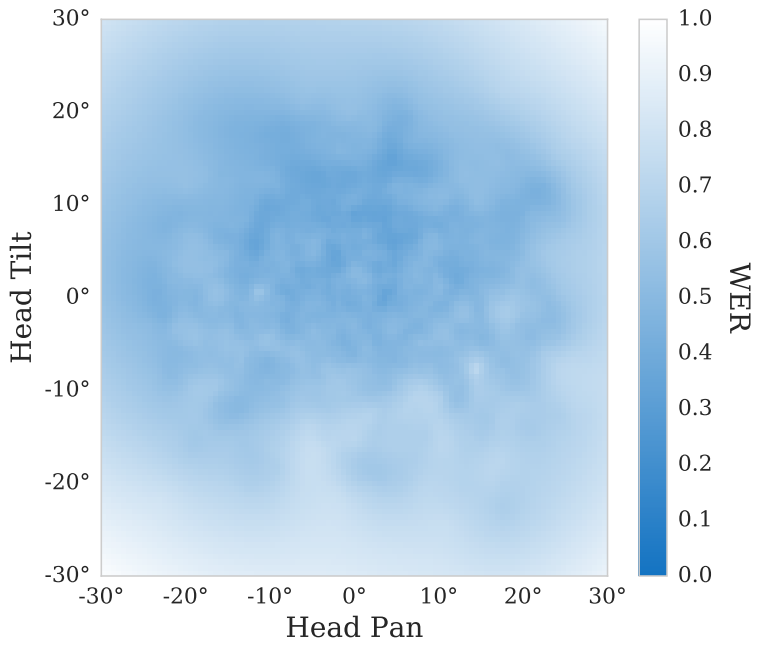


Figure 4.9: Heatmap showing the performance of V2P on different head rotations. Tilt and pan axes are in degrees.

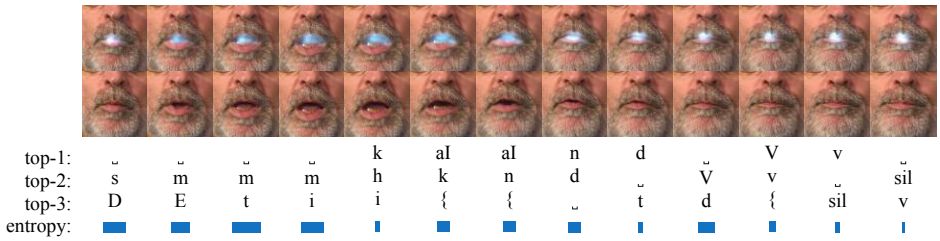
Table 4.6: Evaluation on LRS3-TED.

Model	Filtered Test	Full Test
TM-seq2seq	–	57.9
V2P	$\mathbf{47.0 \pm 1.6}$	$\mathbf{55.1 \pm 0.9}$

4.6.6 Transfer evaluation on LRS3-TED

To demonstrate the generalization power of our V2P approach, we also compare it to the results of the TM-seq2seq model of Afouras et al. (2018a) on LRS3-TED (Afouras et al., 2018b). Unlike LSVSR, the LRS3-TED dataset includes faces at angles between ±90° instead of ±30°, and clips may be shorter than one second. Despite the fact that we do not train or fine-tune V2P on LRS3-TED, our approach still outperforms the state-of-the-art model trained on that dataset in terms of test set accuracy.

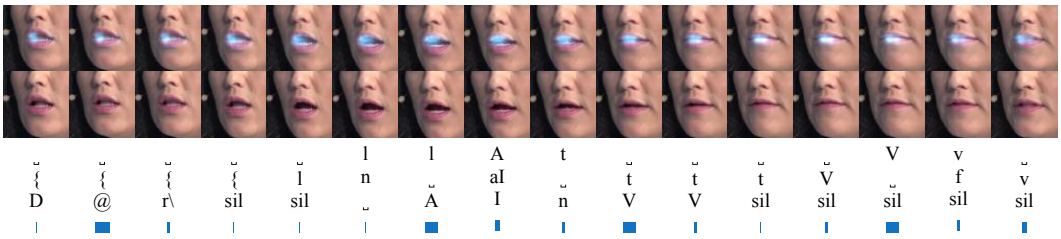
In particular, we conducted two experiments. First, we evaluated performance on a subset of the LRS3-TED test set filtered according to the same protocol used to construct LSVSR, by removing instances with larger face angles and shorter clips (*Filtered Test*). Second, we tested on the full unfiltered test set (*Full Test*). In both cases, V2P outperforms TM-seq2seq, achieving WERs of



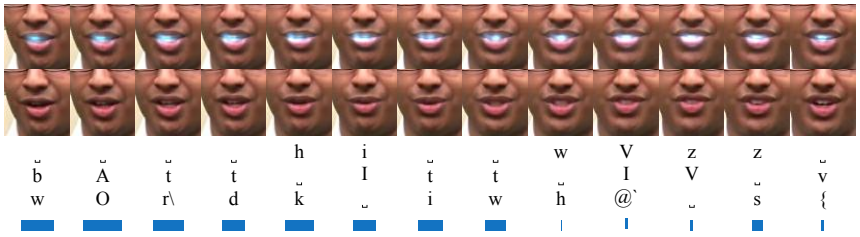
(a) Transcript: "kind of", ground truth phonemes: /k aI n d V v/.



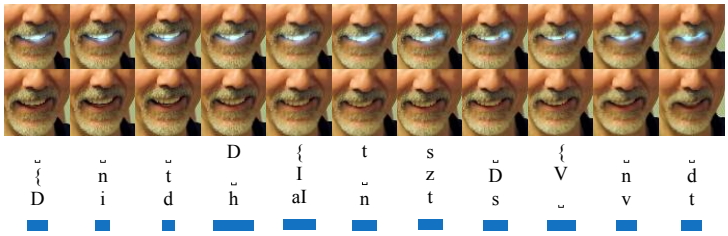
(b) Transcript: "sharing", ground truth phonemes: /S E r\@ N/.



(c) Transcript: "lot of", ground truth phonemes: /l A t V v/.



(d) Transcript: "how was", ground truth phonemes /I t w V z/.



(e) Transcript: "that's a", ground truth phonemes: /D { t s {/.

Figure 4.10: Saliency maps, the top three predictions at each timestep (frame), and the ground truth phonemes. The entropy bar's width represents the entropy of the per-timestep predictive distribution.

47.0 ± 1.6 and 55.1 ± 0.9 respectively. This shows that our approach is able to generalize well, achieving state-of-the-art performance on datasets, with different conditions, on which it was not trained.

4.7 Conclusions

We presented a novel, large-scale visual speech recognition system. Our system consists of a data processing pipeline used to construct a vast dataset—an order of magnitude greater than all previous approaches both in terms of vocabulary and the sheer number of example sequences. We described a scalable model for producing phoneme and word sequences from processed video clips that is capable of nearly halving the error rate of the previous state-of-the-art methods on this dataset, and achieving a new state-of-the-art in a dataset presented contemporaneously with this work. The combination of methods in this work represents a significant improvement in lipreading performance, a technology which can enhance automatic speech recognition systems, and which has enormous potential to improve the lives of speech impaired patients worldwide.

5

Interactive decoding of words from visual speech recognition models

Abstract

This work describes an interactive decoding method to improve the performance of visual speech recognition systems using user input to compensate for the inherent ambiguity of the task. Unlike most phoneme-to-word decoding pipelines, which produce phonemes and feed these through an finite state transducer, our method instead expands words in lockstep, facilitating the insertion of *interaction points* at each word position. Interaction points enable us to solicit input during decoding, allowing users to interactively direct the decoding process. We simulate the behavior of user input using an oracle to give an automated evaluation, and show promise for the use of this method for text input.

5.1 Introduction

Visual speech recognition (VSR) is the task of predicting text from the movement of a user's mouth solely relying on the visual information. VSR has numerous applications ranging from silent speech interfaces to medical applications for people with speech impairments, which is a key motivating factor behind this work. Silent speech interfaces speech recognition in noisy environments, or in places where one does not wish to speak audibly e.g. public transport and open offices. Reliable silent speech interfaces have an enormous potential in helping the lives of hundreds of thousands of patients worldwide. For example, in the U.S. alone, 103,925 tracheostomies were performed in 2014 (HCUPnet, 2014), a procedure that can

result in a difficulty or inability to speak (dysphonia or aphonia). Patients find their loss of voice highly frustrating and because of the difficulty of the task have to rely on professional lipreader services. Since the vocal tract cannot be visually observed as in audio speech recognition, the mapping from phonemes to words is inherently ambiguous. Consequently, the predictive performance as measured by word error rate (WER) is far higher.

Interactively decoding a sequence of words from a word-level model can facilitate inputting correct text despite high uncertainty. This is trivial to implement using beam search in word-level models. However, Chapter 4 showed that phoneme-level connectionist temporal classification (CTC) (Graves et al., 2006) models are more suitable for encoding uncertainties for lipreading compared to non-phoneme CTC approaches. Interactive decoding in this scenario is not straightforward, however, as words correspond to a variable number of phonemes, and CTC marginalizes the alignment of the label sequence to the input, complicating the relationship between time in phoneme-level beam search and the number of resulting words.

In this chapter, we introduce a novel search procedure for decoding words from a phoneme model, synchronized at the word level to permit the insertion of *interaction points*. The *interaction points* pause the decoding procedure, at which one of the shortlisted word candidates is to be selected. This regular narrowing of the search space improves decoding quality. Finally, we measure the performance of our method on a state-of-the-art system called Vision-to-Phoneme (V2P) and evaluate on a real-world dataset.

5.2 Related work

There is a large body of literature on automated lipreading which is extensively outlined in the survey papers of Zhou et al. (2014) and Fernandez-Lopez and Sukno (2018). Earlier work, including those using deep learning, has primarily focused on word classification (e.g. Wand et al., 2016; Stafylakis et al., 2018; Petridis and Pantic, 2016). More recent work increasingly tends to frame lipreading as a sequence prediction task, as in automated speech recognition (e.g. Hinton et al., 2012; Chung and Zisserman, 2016b).

While much work focuses on grapheme-based models (Assael* et al., 2016; Afouras et al., 2018a; Afouras et al., 2018b), we follow the phoneme-based approach of Shillingford* et al. (2019). That work separates the lipreading problem into a phoneme recognition module, trained via CTC loss, and a decoding module using a (weighted) finite state transducer (FST). CTC models the probability of a label

sequence by marginalizing over alignments of the label sequence to the output of a model. Subsequently, the decoder maps the output phoneme sequences to weighted word sequences via a lexicon and language model. Because lipreading is inherently ambiguous due to the vocal tract being visually unobservable, Shillingford* et al. (2019) argue that by partitioning the problem, one can allow the recognition module to model pronunciation uncertainty and the decoder module to separately find likely word sequences consistent with the phoneme distribution, yielding an easier learning problem than having a single model modelling both sources of uncertainty, like in end-to-end grapheme-based models.

In terms of interacting with a model to make predictions, the field of interactive machine translation solves similar problems (Toselli et al., 2011), but usually with a focus on generating a prediction then modifying it and using active learning to learn from these corrections. Closer to our work, Harwath et al. (2014) describe a method for choosing word alternates using n-best lists. Such an approach is intended for displaying a single decoded sequence of words, where each word is associated with a list of alternatives to which they can be changed. In our case, however, we want the user to select an word at *every* position in sequence due to the high ambiguity in lipreading. Thus, to avoid wasted computation for unused word candidates, we only decode up to each word position, and only continue once a word is selected.

5.3 Method

In this section, we describe how to perform a word-level search using an underlyingly phoneme-level CTC model. Recall that our goal is to add an *interaction point* after every word, at which a single word from a list of candidates is to be selected. At a high level, we expand phoneme sequences recursively until every sequence of phonemes corresponds to a full word. At this point, we produce an interaction point. Then, we repeat this procedure, producing one word at a time.

There are several challenges to realizing such an algorithm. First, there are exponentially many sequences of phonemes to which we can expand; we solve this by periodically pruning the set of search states. Second, the underlying phoneme model uses CTC, so there is not a single alignment of a phoneme sequence to the model output. For this, we borrow ideas from CTC beam search that provide a rough approximation of marginalization (Graves and Jaitly, 2014). Finally, and crucially, different words have different numbers of phonemes, and furthermore these phonemes can be output anywhere in the recognition model. Thus, to achieve

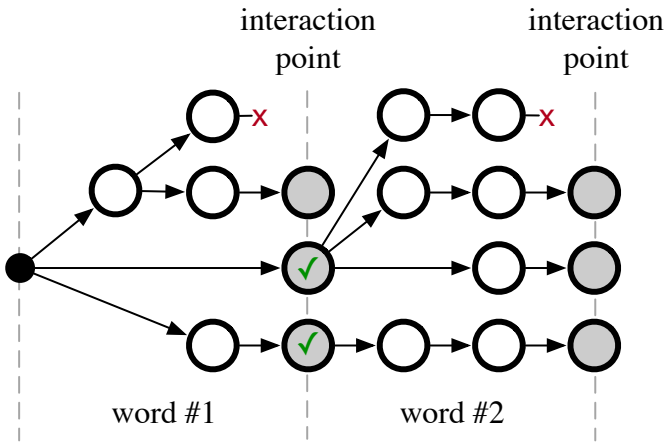


Figure 5.1: Search is expanded at the CTC timestep level until producing enough phonemes (O) such that the decoder FST completely outputs a word. Child search states are formed by expanding by phonemes, and for each phoneme potential branching in the FST. Some transitions lead to a dead end or get pruned due to a low score (×). Upon outputting a word, a search state is frozen (●) until all other search states are also frozen—due to freezing, search states may be in different CTC timesteps. Once all are frozen, an interaction point prompts for the selection of a word (✓), and search states containing other words in this position are pruned. Then repeat to select the next word.

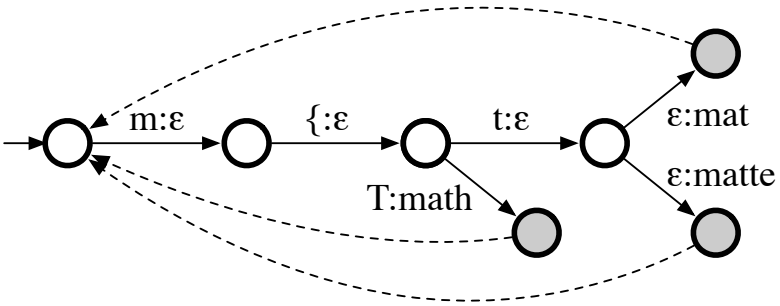


Figure 5.2: A sample decoder FST maps pronunciations to the words 'mat', 'matte' and 'math'. Gray nodes denote accepting states.

word synchrony, unlike a standard CTC beam search, we allow different search states to stop at different timesteps of the model's CTC probabilities.

Another important ingredient to the approach here is the (weighted) FST representing a relation of phoneme strings and word strings. FSTs are non-deterministic finite-state automata that input and output a symbol (possibly epsilon) at each arc. Each arc and accepting (final) state is associated with a weight, together determining a path's weight. We construct a *decoder FST* $L \circ G$, where lexicon L maps from each alternate pronunciation of a word to the word itself, and G is a language model. See Mohri et al. (2002) for an overview of FSTs.

The search state keeps track of the phoneme sequence, the time index into the CTC per-frame label probabilities t_{CTC}, the FST state, the words output so far,

and the cumulative FST path weight. We also keep track of quantities required for correctly computing CTC probabilities.

Algorithm 1 Interactive decoding for VSR. This procedure expands phoneme sequences recursively (with pruning), feeding these through the FST, until every sequence of phonemes has produced a new word. The set of words, ranked by their combined phoneme and FST (i.e. language model) score, is provided to the user (or the automated evaluation procedure of Section 5.4), and the user selects one. See main text for details.

input: decoder FST, `fst`; per-frame label probabilities, Φ.
▷ root search state: $t_{CTC} = 0$, empty phoneme and word strings, initial FST state and path weight = 0, non-frozen.
`fringe` ← [root search state]
$\phi \leftarrow \{(\texttt{phonemes} = \text{``}, t_{CTC} = 0) \rightarrow (p_b = 1, p_{nb} = 0)\}$
`result` ← []
for each word position $w = 1, 2, \ldots$ **do**
  while non-frozen search state exists in `fringe` **do**
    (`fringe`, ϕ) ← ExpandFringe(`fringe`, ϕ, Φ, `fst`)
  From `fringe`, build word `candidates` list sorted by score
  `word`, `stop` ← InteractionPoint(`candidates`)
  if `stop` **then break**
  `result` ← `result` + `word`
  Keep `fringe` search states with `word` at position w

The outermost loop of the search procedure (Algorithm 1) iterates over word positions. The first iteration thus finds a set of candidates for the first word. At the end of this loop, all search states will be in an FST accepting state and will have output exactly one word, as illustrated in Figure 5.1. Finally, all search states that do not end in the selected word are removed. Thus, at the beginning and ending of the outermost loop, all search states contain identical word sequences, but potentially different phoneme sequences, t_{CTC}, and FST states.

Inside this loop, we expand each search state in the *fringe* by one CTC timestep in a round-robin fashion (Algorithm 2). Each search state is extended by every possible phoneme and advanced by one CTC timestep, and also left with its phoneme string untouched and advanced by one CTC timestep. For each of these extensions, zero or more FST states will be produced; one search state is produced for each. If the expansion of a search state results in a word being output, it is *frozen* and no longer participates in expansions. Once the whole fringe is frozen, we stop expanding and insert an interaction point.

As with CTC beam search, we keep track of two quantities for a phoneme sequence at time t_{CTC}: p_b, the probability of the model outputting the phoneme sequence in t_{CTC} timesteps and the last timestep outputting a blank symbol, and

Algorithm 2 Expand fringe by one CTC timestep. This procedure steps each non-frozen search state by one CTC timestep, some ways of which will result a new phoneme. If so, this phoneme along with each of the current FST states are input to the FST to get a new set of states.

```
▷ Expands each non-frozen search state by one CTC timestep, updating CTC phoneme
probabilities stored in φ.
function ExpandFringe(fringe, φ, Φ, fst)
  fringe′ ← {}
  for each non-frozen state in fringe do
    phonemes left as-is, advance t_CTC, update φ using Φ
    for each phoneme p do
      Extend phonemes by p advance t_CTC,
        update φ using Φ, add to fringe′
      new_fst_states ← Feed p into fst and
        stop at accepting states
      for each fst_state in new_fst_states do
        Construct new search state, new_search_state
        if fst_state is accepting state then
          freeze new new_search_state
        Add new new_search_state to fringe′
  return (k best search states in fringe′), φ
```

p_{nb}, the probability of the model outputting the phoneme sequence in t_{CTC} timesteps and the last timestep outputting the final phoneme in the sequence (not blank). The probability of observing a phoneme sequence up to t_{CTC} is $p_b + p_{nb}$. The update formulas computing (p_b, p_{nb}) at timestep $t_{CTC} + 1$ from t_{CTC} follow the same equations as presented in Graves and Jaitly (2014), which can be thought of as a rough approximation to marginalizing out the alignment. As multiple search states may share the same phoneme sequence, we store these using a hashtable keyed by (phoneme sequence, t_{CTC}). For numerical stability, all operations are performed with log probabilities.

After each round-robin expansion of the search states, we prune to retain a constant number of best search states, as in beam search. The score of a search state, used for ranking and pruning, is defined as $-\log(p_b + p_{nb}) + \text{FST path weight}$.

5.4 Evaluation

We use the V2P CTC model from Chapter 4. We then construct a small decoder FST with a word-level bigram language model with Kneser-Ney smoothing and a vocabulary of 10,000 words (Ney et al., 1994). The language model was trained on the training set transcripts in LSVSR. As we only want to generate word candidate

lists, and due to experimental time constraints, we only show results on a small language model. A larger language model could improve the positional ranking of the correct word. We correspondingly subset the test set of Shillingford* et al. (2019) to only the utterances that fit in this vocabulary. We give the top 100 candidates at each interaction point, and expand up to 20 FST states per phoneme, and prune the fringe to the top 200 after expansion.

We define an *oracle* that, at each interaction point, decides which word to select based on the ground-truth transcript. The oracle is defined to mimic a human inputting text and can take 3 actions. If the word candidate list contains the current word in the transcript, the oracle selects it (FOUND CURRENT). If the list contains the next word but not the current, the oracle selects that (FOUND NEXT). If neither the current nor the next word exist in the candidate list, the oracle picks the best scoring one (NOT FOUND). The last strategy is necessary to allow the search procedure to continue even if the correct word is not present. The oracle terminates when no words remain in the transcript.

To evaluate the method, we count the number of times each oracle action is executed. The counts and proportions of success are shown in Section 5.4. The oracle successfully selects a word from the transcript 82.3% of the time. We also measure the number of times the oracle selects a word *besides* the first one in the candidate list, suggesting the utility of interaction points and selecting from a candidate list instead of merely always picking the highest-probability word (SUCCESS RATE EXCL. FIRST).

Oracle action	**Count**	**%**
Not found	410	14.4%
Found current	2097	73.4%
Found next	254	8.9%
Success rate excl. first	1514	53.0%
Success rate	**2351**	**82.3%**

Table 5.1: Oracle actions, across 2857 interaction points. Of these, in 96 cases not counted, the last word in the utterance was not found.

We also measure the distribution of rank of the candidate words. That is, when a word is selected, at what position in the candidate list did it occur?

Finally, we also measure the word error rate (WER) of the predictions produced with interactive decoding using the oracle, and compare against regular decoding using the same decoder FST. For reference, we also include the performance of a large 5-gram LM (as described in Section 4.6) on the same data.

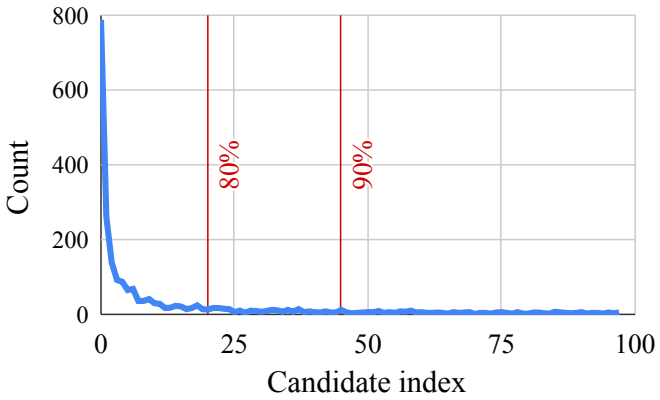


Figure 5.3: Distribution of selected candidate indices.

Decoding method	WER
Standard (10k bigram LM)	62.4%
Standard (5-gram LM)	44.5%
Interactive (10k bigram LM)	33.9%

Table 5.2: WER comparison, 'Interactive' denotes interactive decoding with the oracle, and 'Standard' denotes CTC beam search with the FST.

5.5 Conclusions

We presented a novel method for interactively decoding word sequences from visual speech recognition models. Specifically, we described how to perform a word-level search from a phoneme-level CTC model, i.e. where the outer loop is over word positions, and used this to insert interaction points after each word. We performed an automated evaluation of this procedure, which showed its promise for use in silent speech input applications.

Future work improving the quality of these results includes scaling to a larger language model, and incorporating context *following* the word we are about to select. Furthermore, it is useful to automatically decide when an interaction point is necessary and when we are sufficiently confident that the top word is correct. Initial results showed that thresholding the score gap between the best and second-best word candidate works well. Finally, as emphasized in this work, word-level synchronization is a useful property for interactive decoding. This nested beam search for word-synchronized decoding from phoneme-level models can also be applied to decoding words from grapheme-level or wordpiece-level models.

6 Conclusions

This thesis demonstrated how techniques from speech recognition can be advantageous in a variety of other problems.

In Chapter 2, we solved the problem of filling in missing characters in modern Hawaiian orthography using no paired data whatsoever, by combining a handcrafted component to generate modern Hawaiian text candidates with a language model to disambiguate between these. Without such an approach, this problem is unlikely to have been solvable; our framework provided a convenient means through which the expert knowledge could be encoded. Less convenient alternatives include constructing synthetic data, for instance, but this does not eschew the need for a handcrafted component, and is far less convenient as modification of the handcrafted portion requires retraining.

In Chapter 3, we applied a connectionist temporal classification to the lipreading problem, eliminating the requirement for alignments and thus achieving sentence-level training on lipreading for the first time (at the time of publication). We also showed the importance of jointly learning spatial and temporal features, facilitated by spatiotemporal convolutions.

In Chapter 4, we tackled the problem of large-scale visual speech recognition. In addition to confirming the findings of Chapter 3 on a large dataset we created, we showed that the separation of phoneme recognition and decoding common in audio speech recognition is especially advantageous for lipreading.

In Chapter 5, we looked at an interactive decoding algorithm useful for constructing an interactive lipreading-based text input system. Framing this a user-guided search over the space of words consistent with the phoneme recognition model, as

opposed to learning a network to produce word candidates, allowed for the reuse of the existing visual speech model introduced in Chapter 4, which is prohibitively expensive to repeatedly train. Furthermore, an end-to-end approach for producing the sequences of word candidates is not obvious, and would likely involve pushing the search problem to training time and/or inside the model, neither of which is feasible given the size of the LSVSR model.

As suggested in the introduction, the techniques used in speech recognition are useful for solving other problems in two important ways that we saw in this thesis. The first is to exploit monotonicity, which occurs widely across sequence modeling problems besides just in speech. The second is the arbitrary combination of handcrafted FSTs with trained ones, as a way of encoding expert knowledge into a larger system. This continues to play an important role in speech recognition when data is not very plentiful. We showed that these techniques are useful for other problems that have similar challenges, especially beneficial when paired data for learning the end-to-end problem from scratch is less plentiful or unavailable.

Bibliography

Afouras, T., J. S. Chung, A. Senior, O. Vinyals, and A. Zisserman (2018a). Deep Audio-Visual Speech Recognition. *arXiv preprint arXiv:1809.02108*.

Afouras, T., J. S. Chung, and A. Zisserman (2018b). LRS3-TED: a large-scale dataset for visual speech recognition. *arXiv preprint arXiv:1809.00496*.

Afouras, T., J. S. Chung, and A. Zisserman (2018c). The Conversation: Deep Audio-Visual Speech Enhancement. *arXiv preprint arXiv:1804.04121*.

Akbari, H., H. Arora, L. Cao, and N. Mesgarani (2017). Lip2AudSpec: Speech reconstruction from silent lip movements video. *arXiv preprint arXiv:1710.09798*.

Allauzen, C., M. Riley, and J. Schalkwyk (2010). Filters for efficient composition of weighted finite-state transducers. *International Conference on Implementation and Application of Automata*. Springer, pp. 28–38.

Allauzen, C., M. Riley, J. Schalkwyk, W. Skut, and M. Mohri (2007). OpenFst: A general and efficient weighted finite-state transducer library. *International Conference on Implementation and Application of Automata*. Springer, pp. 11–23.

Almajai, I., S. Cox, R. Harvey, and Y. Lan (2016). Improved speaker independent lip reading using speaker adaptive training and deep neural networks. *International Conference on Acoustics, Speech, and Signal Processing*. IEEE, pp. 2722–2726.

Amodei, D., S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen, et al. (2016). Deep speech 2: End-to-end speech recognition in english and mandarin. *International Conference on Machine Learning*, pp. 173–182.

Andrychowicz, M., M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas (2016). Learning to learn by gradient descent by gradient descent. *Advances in Neural Information Processing Systems*, pp. 3981–3989.

Arjovsky, M., A. Shah, and Y. Bengio (2016). Unitary evolution recurrent neural networks. *International Conference on Machine Learning*, pp. 1120–1128.

Ashby, P. (2013). *Understanding phonetics*. Routledge.

Assael*, Y., B. Shillingford*, S. Whiteson, and N. de Freitas (2016). LipNet: Sentence-level Lipreading. *arXiv preprint arXiv:1611.01599*.

Bahdanau, D., K. Cho, and Y. Bengio (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bear, H. L. and R. Harvey (2016). Decoding visemes: Improving machine lip-reading. *International Conference on Acoustics, Speech, and Signal Processing*. IEEE, pp. 2009–2013.

Bear, H. L. and R. Harvey (2017). Phoneme-to-viseme mappings: the good, the bad, and the ugly. *Speech Communication* 95, pp. 40–67.

Bengio, Y., P. Simard, P. Frasconi, et al. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks* 5.2, pp. 157–166.

Benton, R. A. (1981). *The flight of the Amokura: Oceanic languages and formal education in the South Pacific*. Wellington: New Zealand Council for Educational Research.

Bottou, L. and P. Gallinari (1991). A framework for the cooperation of learning algorithms. *Advances in neural information processing systems*, pp. 781–788.

Chambers, C., A. Raniwala, F. Perry, S. Adams, R. R. Henry, R. Bradshaw, and N. Weizenbaum (2010). FlumeJava: easy, efficient data-parallel pipelines. *Sigplan Notices*. Vol. 45. ACM, pp. 363–375.

Chamisso, A. v. (1837). *Über die Hawaiische Sprache, Vorgelegt der Königlichen Academie der Wissenschaften zu Berlin am 12, Januar, 1837*. Leipzig: Weidmann.

Chen, Y., Y. M. Assael, B. Shillingford, D. Budden, S. E. Reed, H. Zen, Q. Wang, L. C. Cobo, A. Trask, B. Laurie, Ç. Gülçehre, A. van den Oord, O. Vinyals, and N. de Freitas (2018). Sample Efficient Adaptive Text-to-Speech. *arXiv preprint arXiv:1809.10460*.

Cho, K., B. Van Merriënboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Chu, S. M. and T. S. Huang (2000). Bimodal speech recognition using coupled hidden Markov models. *Interspeech*.

Chung, J., A. Senior, O. Vinyals, and A. Zisserman (2017). Lip Reading Sentences in the Wild. *Conference on Computer Vision and Pattern Recognition*.

Chung, J. S. and A. Zisserman (2016a). Out of time: automated lip sync in the wild. *Workshop on Multi-view Lip-reading, ACCV*.

Chung, J. S. and A. Zisserman (2017). Lip reading in profile. *British Machine Vision Conference*.

Chung, J. and A. Zisserman (2016b). Lip Reading in the Wild. *Asian Conference on Computer Vision*.

Chung, J., Ç. Gülçehre, K. Cho, and Y. Bengio (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Cooke, M., J. Barker, S. Cunningham, and X. Shao (2006). An audio-visual corpus for speech perception and automatic speech recognition. *The Journal of the Acoustical Society of America* 120.5, pp. 2421–2424.

Costa, R. P., I. A. Assael, B. Shillingford, N. de Freitas, and T. Vogels (2017). Cortical microcircuits as gated-recurrent neural networks. *Advances in Neural Information Processing Systems*, pp. 272–283.

Cruttenden, A. (2014). *Gimson's pronunciation of English*. Routledge, p. 273.

Dahl, G. E., D. Yu, L. Deng, and A. Acero (2012). Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing* 20.1, pp. 30–42.

DeLand, F. (1931). The story of lip-reading, its genesis and development.

Easton, R. D. and M. Basala (1982). Perceptual dominance during lipreading. *Perception & Psychophysics* 32.6, pp. 562–570.

Eisner, J. (2016). Inside-outside and forward-backward algorithms are just backprop (tutorial paper). *Proceedings of the Workshop on Structured Prediction for NLP*, pp. 1–17.

Elbert, S. H. and M. K. Pukui (1979). *Hawaiian Grammar*. Honolulu: University of Hawai‘i Press.

Elman, J. L. (1990). Finding structure in time. *Cognitive science* 14.2, pp. 179–211.

Ephrat, A., I. Mosseri, O. Lang, T. Dekel, K. Wilson, A. Hassidim, W. T. Freeman, and M. Rubinstein (2018). Looking to Listen at the Cocktail Party: A Speaker-Independent Audio-Visual Model for Speech Separation. *arXiv preprint arXiv:1804.03619*.

Ephrat, A. and S. Peleg (2017). Vid2speech: Speech Reconstruction from Silent Video. *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*.

Fernandez-Lopez, A. and F. Sukno (2018). Survey on automatic lip-reading in the era of deep learning. *Image and Vision Computing*.

Ferragne, E. and F. Pellegrino (2010). Formant frequencies of vowels in 13 accents of the British Isles. *Journal of the International Phonetic Association* 40.01, pp. 1–34.

Fisher, C. G. (1968). Confusions among visually perceived consonants. *Journal of Speech, Language, and Hearing Research* 11.4, pp. 796–804.

Fu, M. C. (2006). Gradient estimation. *Handbooks in operations research and management science* 13, pp. 575–616.

Fu, Y., S. Yan, and T. S. Huang (2008). Classification and feature extraction by simplexization. *IEEE Transactions on Information Forensics and Security* 3.1, pp. 91–100.

Gabbay, A., A. Shamir, and S. Peleg (2017). Visual Speech Enhancement using Noise-Invariant Training. *arXiv preprint arXiv:1711.08789*.

Gales, M., S. Young, et al. (2008). The application of hidden Markov models in speech recognition. *Foundations and Trends® in Signal Processing* 1.3, pp. 195–304.

Garg, A., J. Noyola, and S. Bagadia (2016). *Lip reading using CNN and LSTM*. Tech. rep. Stanford University, CS231n project report.

Gergen, S., S. Zeiler, A. H. Abdelaziz, R. Nickel, and D. Kolossa (2016). Dynamic Stream Weighting for Turbo-Decoding-Based Audiovisual ASR. *Interspeech*, pp. 2135–2139.

Gers, F. A., J. Schmidhuber, and F. Cummins (2000). Learning to forget: Continual prediction with LSTM. *Neural computation* 12.10, pp. 2451–2471.

Goldschen, A. J., O. N. Garcia, and E. D. Petajan (1997). Continuous automatic speech recognition by lipreading. *Motion-Based recognition*. Springer, pp. 321–343.

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep learning*.

Graves, A. (2012a). Sequence transduction with recurrent neural networks. *arXiv preprint arXiv:1211.3711*.

Graves, A. (2012b). Supervised sequence labelling. *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer, pp. 5–13.

Graves, A., S. Fernández, F. Gomez, and J. Schmidhuber (2006). Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. *ICML*, pp. 369–376.

Graves, A. and N. Jaitly (2014). Towards End-To-End Speech Recognition with Recurrent Neural Networks. *ICML*, pp. 1764–1772.

Graves, A. and J. Schmidhuber (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks* 18.5, pp. 602–610.

Gray, M. S., J. R. Movellan, and T. J. Sejnowski (1997). Dynamic features for visual speechreading: A systematic comparison. *Advances in Neural Information Processing Systems*, pp. 751–757.

Griewank, A. and A. Walther (2008). *Evaluating derivatives: principles and techniques of algorithmic differentiation*. Vol. 105. Siam.

Gupta, A., B. Shillingford, Y. Assael, and T. C. Walters (2019). Speech Bandwidth Extension with Wavenet. *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pp. 205–208.

Gupta, S., J. Hoffman, and J. Malik (2015). Cross modal distillation for supervision transfer. *arXiv preprint arXiv:1507.00448*.

Gurban, M. and J.-P. Thiran (2009). Information theoretic feature extraction for audio-visual speech recognition. *IEEE Transactions on Signal Processing* 57.12, pp. 4765–4776.

Hajič, J., J. Hric, and V. Kuboň (2000). Machine translation of very close languages. *ANLC '00 Proceedings of the sixth conference on Applied natural language processing*, pp. 7–12.

Hannun, A., C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, et al. (2014). Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*.

Harwath, D., A. Gruenstein, and I. McGraw (2014). Choosing useful word alternates for automatic speech recognition correction interfaces. *Conference of the International Speech Communication Association*.

Hassanat, A. B. (2011). Visual Speech Recognition. *Speech and Language Technologies*. InTech.

HCUPnet (2014). *Hospital Inpatient National Statistics*. `https://hcupnet.ahrq.gov/`. Accessed: 2018-04-23.

He, K., X. Zhang, S. Ren, and J. Sun (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *IEEE International Conference on Computer Vision*, pp. 1026–1034.

He, K., X. Zhang, S. Ren, and J. Sun (2016). Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Henaff, M., A. Szlam, and Y. LeCun (2016). Recurrent orthogonal networks and long-memory tasks. *arXiv preprint arXiv:1602.06662*.

Hilder, S., R. Harvey, and B.-J. Theobald (2009). Comparison of human and machine-based lip-reading. *AVSP*, pp. 86–89.

Hinton, G., L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine* 29.6, pp. 82–97.

Hinton, G., O. Vinyals, and J. Dean (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Hoʻoulumāhiehie (2007). *Ka Moʻolelo o Hiʻiakaikapoliopele*. Edited by M. Puakea Nogelmeier. Honolulu: Awaiaulu Press.

Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02, pp. 107–116.

Hochreiter, S. and J. Schmidhuber (1997). Long short-term memory. *Neural computation* 9.8, pp. 1735–1780.

Hu, D., X. Li, et al. (2016). Temporal multimodal learning in audiovisual speech recognition. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3574–3582.

Ji, S., W. Xu, M. Yang, and K. Yu (2013). 3D convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence* 35.1, pp. 221–231.

Jing, L., Y. Shen, T. Dubcek, J. Peurifoy, S. Skirlo, Y. LeCun, M. Tegmark, and M. Soljačić (2017). Tunable efficient unitary neural networks (EUNN) and their application to RNNs. *International Conference on Machine Learning*, pp. 1733–1741.

Kamakau, S. M. and H. Perreira (2002). Ka Moʻolelo o Kahahana, Māhele 1. *Ka Hoʻoilina* 1.1, pp. 102–121.

Karpathy, A., G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei (2014). Large-scale video classification with convolutional neural networks. *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 1725–1732.

Katz, S. (1987). Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE transactions on acoustics, speech, and signal processing* 35.3, pp. 400–401.

Kim, Y., C. Denton, L. Hoang, and A. M. Rush (2017). Structured attention networks. *arXiv preprint arXiv:1702.00887*.

Kim, Y. and A. M. Rush (2016). Sequence-level knowledge distillation. *arXiv preprint arXiv:1606.07947*.

King, D. E. (2009). Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research* 10.Jul, pp. 1755–1758.

Kingma, D. and J. Ba (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kneser, R. and H. Ney (1995). Improved backing-off for m-gram language modeling. *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. Vol. 1. IEEE, pp. 181–184.

Koller, O., H. Ney, and R. Bowden (2015). Deep Learning of Mouth Shapes for Sign Language. *ICCV Workshop on Assistive Computer Vision and Robotics*, pp. 85–91.

Kolovratník, D., N. Klyueva, and O. Bojar (2010). Statistical machine translation between related and unrelated languages. *Proceedings of the Conference on Theory and Practice of Information Technologies (ITAT-09)*, pp. 31–36.

Koumparoulis, A., G. Potamianos, Y. Mroueh, and S. J. Rennie (2017). Exploring ROI size in deep learning based lipreading. *International Conference on Auditory-Visual Speech Processing*.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). Imagenet classification with deep convolutional neural networks. *NIPS*, pp. 1097–1105.

Kuznetsov, V., H. Liao, M. Mohri, M. Riley, and B. Roark (2016). Learning N-Gram Language Models from Uncertain Data. *Interspeech*, pp. 2323–2327.

Le Cornu, T. and B. Milner (2017). Generating intelligible audio speech from visual speech. *IEEE Transactions on Audio, Speech, and Language Processing* 25.9, pp. 1751–1761.

Le, Q. V., N. Jaitly, and G. E. Hinton (2015). A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*.

Lecouteux, B., G. Linarès, and S. Oger (2012). Integrating imperfect transcripts into speech recognition systems for building high-quality corpora. *Computer Speech & Language* 26.2, pp. 67–89.

LeCun, Y., Y. Bengio, and G. Hinton (2015). Deep learning. *Nature* 521.7553, pp. 436–444.

Liao, H., E. McDermott, and A. Senior (2013). Large scale deep neural network acoustic modeling with semi-supervised training data for YouTube video transcription. *Workshop on Automatic Speech Recognition and Understanding*. IEEE, pp. 368–373.

Liao, H., G. Pundak, O. Siohan, M. K. Carroll, N. Coccaro, Q.-M. Jiang, T. N. Sainath, A. Senior, F. Beaufays, and M. Bacchiani (2015). Large vocabulary automatic speech recognition for children. *ISCA*.

Lucey, P. and S. Sridharan (2006). Patch-based representation of visual speech. *Proceedings of the HCSNet workshop on Use of vision in human-computer interaction*, pp. 79–85.

Maas, A. L., Z. Xie, D. Jurafsky, and A. Y. Ng (2015). Lexicon-free conversational speech recognition with neural networks. *NAACL*.

Makino, T., H. Liao, Y. Assael, B. Shillingford, B. Garcia, O. Braga, and O. Siohan (2019). Recurrent Neural Network Transducer for Audio-Visual Speech Recognition. *Proceedings of IEEE Automatic Speech Recognition and Understanding Workshop*.

Mas, J. and G. Fernandez (2003). Video shot boundary detection based on color histogram. *Notebook Papers TRECVID2003, Gaithersburg, Maryland, NIST* 15.

Mase, K. and A. Pentland (1991). Automatic lipreading by optical-flow analysis. *Systems and Computers in Japan* 22.6, pp. 67–76.

Matthews, I., T. F. Cootes, J. A. Bangham, S. Cox, and R. Harvey (2002). Extraction of visual features for lipreading. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.2, pp. 198–213.

McGraw, I., R. Prabhavalkar, R. Alvarez, M. G. Arenas, K. Rao, D. Rybach, O. Alsharif, H. Sak, A. Gruenstein, F. Beaufays, et al. (2016). Personalized speech recognition on mobile devices. *International Conference on Acoustics, Speech, and Signal Processing*. IEEE, pp. 5955–5959.

McGurk, H. and J. MacDonald (1976). Hearing lips and seeing voices. *Nature* 264, pp. 746–748.

Miao, Y., M. Gowayyed, and F. Metze (2015). EESEN: End-to-end speech recognition using deep RNN models and WFST-based decoding. *Workshop on Automatic Speech Recognition and Understanding*. IEEE, pp. 167–174.

Mikolov, T., M. Karafiát, L. Burget, J. Černocký, and S. Khudanpur (2010a). Recurrent neural network based language model. *Proceedings of Interspeech*. Ed. by T. Kobayashi, K. Hirose, and S. Nakamura, pp. 1045–1048.

Mikolov, T., M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur (2010b). Recurrent neural network based language model. *Eleventh Annual Conference of the International Speech Communication Association*.

Mohri, M. (1997). Finite-state transducers in language and speech processing. *Computational linguistics* 23.2, pp. 269–311.

Mohri, M., F. Pereira, and M. Riley (2002). Weighted finite-state transducers in speech recognition. *Computer Speech & Language* 16.1, pp. 69–88.

Mohri, M., F. Pereira, and M. Riley (2008). Speech recognition with weighted finite-state transducers. *Springer Handbook of Speech Processing*. Springer, pp. 559–584.

Mohri, M. and M. Riley (2002). An efficient algorithm for the n-best-strings problem. *Seventh International Conference on Spoken Language Processing*.

Morris, M. A. and A. N. Kho (2014). Silence in the EHR: Infrequent documentation of aphonia in the electronic health record. *BMC Health Services Research* 14.1, p. 425.

Neti, C., G. Potamianos, J. Luettin, I. Matthews, H. Glotin, D. Vergyri, J. Sison, and A. Mashari (2000). *Audio visual speech recognition*. Tech. rep. IDIAP.

Ney, H., U. Essen, and R. Kneser (1994). On structuring probabilistic dependences in stochastic language modelling. *Computer Speech & Language* 8.1, pp. 1–38.

Ngiam, J., A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng (2011). Multimodal deep learning. *ICML*, pp. 689–696.

Ninomiya, H., N. Kitaoka, S. Tamura, Y. Iribe, and K. Takeda (2015). Integration of deep bottleneck features for audio-visual speech recognition. *International Speech Communication Association*.

Noda, K., Y. Yamaguchi, K. Nakadai, H. G. Okuno, and T. Ogata (2014). Lipreading using convolutional neural network. *INTERSPEECH*, pp. 1149–1153.

Nogelmeier, M. P. (2010). *Mai Pa'a i ka Leo: Historical Voice in Hawaiian Primary Materials: Looking Forward and Listening Back*. Honolulu: Bishop Museum Press.

Östling, R. and J. Tiedemann (2017). Neural machine translation for low-resource languages. *arXiv preprint arXiv:1708.05729*. version 1.

Paleček, K. (2017). Utilizing Lipreading in Large Vocabulary Continuous Speech Recognition. *Speech and Computer*. Springer, pp. 767–776.

Papandreou, G., A. Katsamanis, V. Pitsikalis, and P. Maragos (2007). Multimodal fusion and learning with uncertain features applied to audiovisual speech recognition. *Workshop on Multimedia Signal Processing*, pp. 264–267.

Papandreou, G., A. Katsamanis, V. Pitsikalis, and P. Maragos (2009). Adaptive multimodal fusion by uncertainty compensation with application to audiovisual speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing* 17.3, pp. 423–435.

Parker Jones, O. (2010). A computational phonology and morphology of Hawaiian. PhD thesis. University of Oxford.

Parker Jones, O. (2018). Illustrations of the IPA: Hawaiian. *Journal of the International Phonetic Association* 48, pp. 103–115.

Parker Jones*, O. and B. Shillingford* (2018). Composing RNNs and FSTs for Small Data: Recovering Missing Characters in Old Hawaiian Text. *Interpretability and Robustness in Audio, Speech, and Language Workshop at NeurIPS 2018*.

Pascanu, R., T. Mikolov, and Y. Bengio (2013). On the difficulty of training recurrent neural networks. *International conference on machine learning*, pp. 1310–1318.

Pech-Pacheco, J. L., G. Cristóbal, J. Chamorro-Martinez, and J. Fernández-Valdivia (2000). Diatom autofocusing in brightfield microscopy: A comparative study. *Pattern Recognition*. Vol. 3. IEEE, pp. 314–317.

Petridis, S. and M. Pantic (2016). Deep complementary bottleneck features for visual speech recognition. *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE, pp. 2304–2308.

Petridis, S., Y. Wang, Z. Li, and M. Pantic (2017). End-to-End Multi-View Lipreading. *BMVC*.

Pitsikalis, V., A. Katsamanis, G. Papandreou, and P. Maragos (2006). Adaptive multimodal fusion by uncertainty compensation. *INTERSPEECH*.

Potamianos, G., E. Cosatto, H. P. Graf, and D. B. Roe (1997). Speaker independent audio-visual database for bimodal ASR. *Audio-Visual Speech Processing: Computational & Cognitive Science Approaches*.

Potamianos, G. and H. P. Graf (1998). Discriminative training of HMM stream exponents for audio-visual speech recognition. *International Conference on Acoustics, Speech and Signal Processing*. Vol. 6. IEEE, pp. 3733–3736.

Potamianos, G., H. P. Graf, and E. Cosatto (1998). An image transform approach for HMM based automatic lipreading. *Conference on Image Processing*. IEEE, pp. 173–177.

Potamianos, G., C. Neti, G. Gravier, A. Garg, and A. W. Senior (2003). Recent advances in the automatic recognition of audiovisual speech. *Proceedings of the IEEE* 91.9, pp. 1306–1326.

Potamianos, G., C. Neti, J. Luettin, and I. Matthews (2004). Audio-visual automatic speech recognition: An overview. *Issues in Visual and Audio-Visual Speech Processing* 22, p. 23.

Prabhavalkar, R., K. Rao, T. Sainath, B. Li, L. Johnson, and N. Jaitly (2017). A Comparison of Sequence-to-Sequence Models for Speech Recognition. *Interspeech*.

Pukui, M. K. and S. H. Elbert (1957). *Hawaiian-English Dictionary*. Honolulu: University of Hawai‘i Press.

Pukui, M. K., H. Ho‘opai, O. Parker Jones, and K. NeSmith (2006). No ka mahi‘ai ‘ana, Māhele 6. *Ka Ho‘oilina* 5.1, pp. 2–23.

Rao, K., H. Sak, and R. Prabhavalkar (2017). Exploring architectures, data and units for streaming end-to-end speech recognition with RNN-transducer. *Automatic Speech Recognition and Understanding Workshop (ASRU), 2017 IEEE*. IEEE, pp. 193–199.

Rastogi, P., R. Cotterell, and J. Eisner (2016). Weighted finite-state transductions with neural context. *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 623–633.

Roark, B., R. Sproat, C. Allauzen, M. Riley, J. Sorensen, and T. Tai (2012). The OpenGrm open-source finite-state grammar software libraries. *Proceedings of the ACL 2012 System Demonstrations*, pp. 61–66.

Sagonas, C., G. Tzimiropoulos, S. Zafeiriou, and M. Pantic (2013). 300 faces in-the-wild challenge: The first facial landmark localization challenge. *IEEE International Conference on Computer Vision Workshops*, pp. 397–403.

Saitoh, T. and R. Konishi (2010). A study of influence of word lip reading by change of frame rate. *Auditory-Visual Speech Processing 2010*.

Salcianu, A., A. Golding, A. Bakalov, C. Alberti, D. Andor, D. Weiss, E. Pitler, G. Coppola, J. Riesa, K. Ganchev, M. Ringgaard, N. Hua, R. McDonald, S. Petrov, S. Istrate, and T. Koo (2018). *Compact Language Detector v3*. `https://github.com/google/cld3`.

Saxe, A. M., J. L. McClelland, and S. Ganguli (2013). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*.

Scannell, K. (2014). Statistical models for text normalization and machine translation. *Proceedings of the First Celtic Language Technology Workshop*, pp. 33–40.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks* 61, pp. 85–117.

Schroff, F., D. Kalenichenko, and J. Philbin (2015). Facenet: A unified embedding for face recognition and clustering. *Conference on Computer Vision and Pattern Recognition*, pp. 815–823.

Schütz, A. J. (1994). *The Voices of Eden: A history of Hawaiian language studies*. Honolulu: University of Hawai'i Press.

Shaikh, A. A., D. K. Kumar, W. C. Yau, M. C. Azemin, and J. Gubbi (2010). Lip reading using optical flow and support vector machines. *International Congress on Image and Signal Processing*. Vol. 1. IEEE, pp. 327–330.

Shillingford, B., Y. Assael, and M. Denil (2019). Interactive decoding of words from visual speech recognition models. *In submission*.

Shillingford*, B., Y. Assael*, M. W. Hoffman, T. Paine, C. Hughes, U. Prabhu, H. Liao, H. Sak, K. Rao, L. Bennett, M. Mulville, M. Denil, B. Coppin, B. Laurie, A. Senior, and N. de Freitas (2018). Large-scale visual speech recognition. *arXiv preprint arXiv:1807.05162*.

Shillingford*, B., Y. Assael*, M. W. Hoffman, T. Paine, C. Hughes, U. Prabhu, H. Liao, H. Sak, K. Rao, L. Bennett, M. Mulville, M. Denil, B. Coppin, B. Laurie, A. Senior, and N. de Freitas (2019). Large-scale visual speech recognition. *Proc. Interspeech 2019*, pp. 4135–4139.

Shillingford*, B. and O. Parker Jones* (2018). Recovering missing characters in old Hawaiian writing. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4929–4934.

Simonyan, K., A. Vedaldi, and A. Zisserman (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*.

Simonyan, K. and A. Zisserman (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. *ICRL*.

Sipser, M. (2012). *Introduction to the Theory of Computation*. Cengage learning.

Soltau, H., H. Liao, and H. Sak (2017). Neural speech recognizer: Acoustic-to-word LSTM model for large vocabulary speech recognition. *Interspeech*, pp. 3707–3711.

Springenberg, J. T., A. Dosovitskiy, T. Brox, and M. Riedmiller (2014). Striving for simplicity: The all convolutional net. *ICLR Workshop*.

Sproat, R. and N. Jaitly (2016). RNN approaches to text normalization: A challenge. *arXiv preprint arXiv:1611.00068*. version 2.

Stafylakis, T., M. H. Khan, and G. Tzimiropoulos (2018). Pushing the boundaries of audiovisual word recognition using Residual Networks and LSTMs. *Computer Vision and Image Understanding* 176-177, pp. 22–32.

Stafylakis, T. and G. Tzimiropoulos (2017). Combining Residual Networks with LSTMs for Lipreading. *Interspeech*. ISCA, pp. 3652–3656.

Sui, C., M. Bennamoun, and R. Togneri (2015). Listening With Your Eyes: Towards a Practical Visual Speech Recognition System Using Deep Boltzmann Machines. *IEEE International Conference on Computer Vision*, pp. 154–162.

Sutskever, I., O. Vinyals, and Q. V. Le (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, pp. 3104–3112.

Takashima, Y., R. Aihara, T. Takiguchi, Y. Ariki, N. Mitani, K. Omori, and K. Nakazono (2016). Audio-Visual Speech Recognition Using Bimodal-Trained Bottleneck Features for a Person with Severe Hearing Loss. *INTERSPEECH*, pp. 277–281.

Tamura, S., K. Iwano, and S. Furui (2004). Multi-modal speech recognition using optical-flow analysis for lip images. *Real World Speech Processing*. Springer, pp. 43–50.

Taylor, S., B.-J. Theobald, and I. Matthews (2014). The effect of speaking rate on audio and visual speech. *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE, pp. 3037–3041.

Thanda, A. and S. M. Venkatesan (2017). Audio Visual Speech Recognition Using Deep Recurrent Neural Networks. *Multimodal Pattern Recognition of Social Signals in Human-Computer-Interaction*. Ed. by F. Schwenker and S. Scherer. Springer, pp. 98–109.

The Health Foundation (2014). *New safety collaborative will improve outcomes for patients with tracheostomies*. `http://www.health.org.uk/news/new-safety-collaborative-will-improve-outcomes-patients-tracheostomies`. Accessed: 2018-04-23.

Torfi, A., S. M. Iranmanesh, N. Nasrabadi, and J. Dawson (2017). 3D Convolutional Neural Networks for Cross Audio-Visual Matching Recognition. *IEEE Access* 5, pp. 22081–22091.

Toselli, A. H., E. Vidal, and F. Casacuberta (2011). Interactive machine translation. *Multimodal Interactive Pattern Recognition and Applications*. Springer, pp. 135–152.

Wand, M., J. Koutnik, and J. Schmidhuber (2016). Lipreading with long short-term memory. *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 6115–6119.

Wand, M. and J. Schmidhuber (2017). Improving Speaker-Independent Lipreading with Domain-Adversarial Training. *Interspeech*.

Wang, S.-L., A. W.-C. Liew, W. H. Lau, and S. H. Leung (2008). An automatic lipreading system for spoken digits with limited training data. *IEEE Transactions on Circuits and Systems for Video Technology* 18.12, pp. 1760–1765.

Warner, S. L. N. (2001). The Movement to Revitalize Hawaiian Language and Culture. *The Green Book of Language Revitalization in Practice*. Ed. by L. Hinton and K. Hale. San Diego, CA: Academic Press, pp. –144.

Wells, J. C. (1995). *Computer-coding the IPA: A proposed extension of SAMPA*. Tech. rep.

Wilson, W. H. (1976). *Standardized Hawaiian orthography*. Manuscript, University of Hawai‘i.

Wilson, W. H. and K. Kamanā (2001). “Mai Loko Mai o ka ‘I‘ini: Proceeding from a Dream”: The ‘Aha Pūnana Leo Connection in Hawaiian Language Revitalization. *The Green Book of Language Revitalization in Practice*. Ed. by L. Hinton and K. Hale. San Diego, CA: Academic Press, pp. 147–176.

Woodward, M. F. and C. G. Barber (1960). Phoneme perception in lipreading. *Journal of Speech, Language, and Hearing Research* 3.3, pp. 212–222.

Wu, Y. and K. He (2018). Group Normalization. *arXiv preprint arXiv:1803.08494*.

Xu, K., D. Li, N. Cassimatis, and X. Wang (2018). LCANet: End-to-End Lipreading with Cascaded Attention-CTC. *arXiv preprint arXiv:1803.04988*.

Yoshinaga, T., S. Tamura, K. Iwano, and S. Furui (2003). Audio-visual speech recognition using lip movement extracted from side-face images. *International Conference on Audio-Visual Speech Processing*.

Yu, L., J. Buys, and P. Blunsom (2016). Online segment to segment neural transduction. *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Zeiler, M. D. and R. Fergus (2014). Visualizing and understanding convolutional networks. *European Conference on Computer Vision*, pp. 818–833.

Zhao, G., M. Barnard, and M. Pietikainen (2009). Lipreading with local spatiotemporal descriptors. *IEEE Transactions on Multimedia* 11.7, pp. 1254–1265.

Zheng, F., G. Zhang, and Z. Song (2001). Comparison of different implementations of MFCC. *Journal of Computer science and Technology* 16.6, pp. 582–589.

Zhou, Z., G. Zhao, X. Hong, and M. Pietikäinen (2014). A review of recent advances in visual speech decoding. *Image and Vision Computing* 32.9, pp. 590–605.

Appendices

A

Sample predictions for Chapter 2

This appendix contains sample predictions of the model from Chapter 2: *Restoring Hawaiian orthography using FSTs and a hybrid FST/RNN approach*, on the first 10 sentences of each newspaper excerpts described in the chapter.

Each block of three lines begins with the input, followed by the prediction, followed by the ground-truth. We denote characters omitted by the model as compared to the ground-truth by *blue and italic*, whereas characters that are erroneously inserted or substituted (or should have been substituted) for another character are marked with red underline. When the incorrect character is a space, the space is replaced with '␣'. Note that correct changes are left unmarked.

First 10 sentences in Newspaper 1:

Input 1: Weheia ka Malapua Alii a Kanuia na Uluwehi no ia Wao.
Prediction 1: Wehe ʻia ka Māla pua Aliʻi a Kanu␣ʻia na Uluwehi nō ia Wao.
Ground-truth 1: Wehe ʻia ka Māla Pua Aliʻi a Kanu ʻia nā Uluwehi no ia Wao.

Input 2: E like no hoi me ka mea i hoike akea ia ae no ka manawa a me ka la e weheia ai a e kanuia ai hoi o na pua a me na mea ulu e ae ma kahi nona ka inoa kilakila maluna ae, pela no i hoea io mai ai i ka Poaha iho la, hora 9 A. M. a mahope mai. Mamua ae o ia manawa, ua lehulehu na poe i pii aku me na mea kanu, maluna o na kaa a malalo no hoi.
Prediction 2: E like nō hoʻi me ka mea i hōʻike ākea ʻia aʻe no ka manawa a me ka lā e wehe ʻia ai a e kanu ʻia ai hoʻi ʻo nā pua a me nā mea ulu ʻē aʻe ma kahi nona ka inoa kilakila ma luna aʻe, pēlā nō i hōʻea ʻiʻo mai ai i ka Poʻahā iho la,

hora 9 A. M. a ma hope mai. Ma mua aʻe o ia manawa, ua lehulehu nā poʻe i piʻi aku me nā mea kanu, ma luna o nā kaʻa a ma lalo nō hoʻi.

Ground-truth 2: E like nō hoʻi me ka mea i hōʻike ākea ʻia aʻe no ka manawa a me ka lā e wehe ʻia ai a e kanu ʻia ai hoʻi o nā pua a me nā mea ulu ʻē aʻe ma kahi nona ka inoa kilakila ma luna aʻe, pēlā nō i hōʻea ʻiʻo mai ai i ka Pōʻahā ihola, hola 9 A. M. a ma hope mai. Ma mua aʻe o ia manawa, ua lehulehu nā poʻe i piʻi aku me nā mea kanu, ma luna o nā kaʻa a ma lalo nō hoʻi.

Input 3: Hoomakaia ke Kanu Ana.

Prediction 3: Hoʻomaka ʻia ke Kanu ʻAna.

Ground-truth 3: Hoʻomaka ʻia ke Kanu ʻAna.

Input 4: Ua hoea ae no ilaila ka Puali Puhiohe Lahui, a i ka aneane ana ae i ka manawa, a i ole ia, ua hala no paha he hapalua hora mahope iho o ka hora 9, ua uhene mai la lakou i ke mele Liliuokalani, a o ka wa no ia o Kamalii Kawananakoa, ma ka aoao o ke Aliiaimoku, i kanu iho ai i kekahi kumu lehua o Mokaulele iwaenakonu, i hoopuniia ae me na ohawai a me kekahi mau mea kanu Hawaii e ae iloko o kekahi ponaha poepoe, a makai iho hoi o ia wahi i kanu ai o Kamalii Kalanianaole i kekahi kumu lehua ahihi ma ka aoao o ke Alii ka Moiwahine Kanemake. Pau keia mau hana ae la, ua noa i na mea a pau, a ua hele no hoi ia wahi a eeu i na oiwi palupalu o kakou, e kanu ana i kela a me keia mea, a he mau oiwi oolea no hoi kekahi malaila e kokua ana. He mea makehewa paha ke helupapa aku i na pua i kanuia. Hookahi a makou mea i mahalo, oia no kekahi wahi i kanu mua e ia no makai iki mai o ka puka komo, me ka inoa o ia kihapai e kau ae la maluna, a he ku maoli no i ka nani.

Prediction 4: Ua hōʻea aʻe nō i laila ka Pūʻali Puhi ʻohe Lāhui, a i ka ʻaneʻane ʻana aʻe i ka manawa, a i ʻole ia, ua hala nō paha he hapalua hora ma hope iho o ka hora 9, ua ʻuhene mai la lākou i ke mele Liliʻuokalani, a ʻo ka wā nō ia o Kāmaliʻi Kawānanakoa, ma ka ʻaoʻao o ke Aliʻi ʻai moku, i kanu iho ai i kekahi kumu lehua o Mōkaulele i waenakonu, i hoʻopuni ʻia aʻe me nā ʻohā wai a me kekahi mau mea kanu Hawaiʻi ʻē aʻe i loko o kekahi pōnaha poepoe, a ma kai iho hoʻi o ia wahi i kanu ai ʻo Kamāliʻi Kalanianaʻole i kekahi kumu lehua ʻāhihi ma ka ʻaoʻao o ke Aliʻi, ka Mōʻīwahine Kāne make. Pau kēia mau hana aʻe lā, ua noa i nā mea a pau, a ua hele nō hoʻi ia wahi a ʻe ʻeu i nā ʻōiwi palupalu o kākou, e kanu ana i kēlā a me kēia mea, a he mau ʻōiwi ʻoʻoleʻa nō hoʻi kekahi ma laila e kōkua ana. He mea makehewa paha ke helu papa aku i nā pua i kanu ʻia. Hoʻokahi a mākou

mea i mahalo, ʻo ia nō kekahi wahi i kanu mua ʻē ʻia nō ma kai iki mai o ka puka komo, me ka inoa o ia kīhāpai e kau aʻe lā ma luna, a he kū maoli nō i ka nani.

Ground-truth 4: Ua hōʻea aʻe nō i laila ka pūʻali puhi ʻohe lāhui, a i ka ʻaneʻane ʻana aʻe i ka manawa, a i ʻole ia, ua hala nō paha he hapalua hola ma hope iho o ka hola 9, ua ʻuhene maila lākou i ke mele Liliʻuokalani, a ʻo ka wā nō ia o kamāliʻi Kawānanakoa, ma ka ʻaoʻao o ke aliʻi ʻai moku, i kanu iho ai i kekahi kumu lehua o Mokaulele i waenakonu, i hoʻopuni ʻia aʻe me nā ʻōhāwai a me kekahi mau mea kanu Hawaiʻi ʻē aʻe i loko o kekahi pōnaha poepoe, a ma kai iho hoʻi o ia wahi i kanu ai ʻo kamāliʻi Kalanianaʻole i kekahi kumu lehua ʻāhihi ma ka ʻaoʻao o ke aliʻi, ka mōʻī wahine kāne make. Pau kēia mau hana aʻela, ua noa i nā mea a pau, a ua hele nō hoʻi ia wahi a ʻeʻeu i nā ʻōiwi palupalu o kākou, e kanu ana i kēlā a me kēia mea, a he mau ʻōiwi ʻoʻoleʻa nō hoʻi kekahi ma laila e kōkua ana. He mea makehewa paha ke helu papa aku i nā pua i kanu ʻia. Hoʻokahi a mākou mea i mahalo, ʻo ia nō kekahi wahi i kanu mua e ia nō ma kai iki mai o ka puka komo, me ka inoa o ia kīhāpai e kau aʻela ma luna, a he kū maoli nō i ka nani.

Input 5: Ka Lanai Pea Ahaaina.

Prediction 5: Ka Lānaʻi Peʻa ʻAhaʻaina.

Ground-truth 5: Ka Lānai Peʻa ʻAhaʻaina.

Input 6: Mauka iki aku hoi o keia mala, ma ka aoao maluna o ka hale noho o J. Mana, ua kukuluia ae la he lanai pea, a ilaila kahi i hoomakaukauia ai o kekahi papaaina. E houhene mau ana no hoi ka puali puhiohe i kela a me keia wa a hiki i ka wa ai, i ke kau pono o ka la i ka piko a mahope iho paha, me ka hoomau aku no i ka hoolealea ana a pau ka papaaina mua a ke Alii, o na Kamaliikane no ma kona a me ko kona kaikoeke wahi. Ekolu papaaina i hoonohoia ai a ua ai na poe a pau i hiki aku a lawa pono, me ka hoounaia o kahi mea-ai no kekahi poe i maopopo i hiki ole aku.

Prediction 6: Ma uka iki aku hoʻi o kēia māla, ma ka ʻaoʻao ma luna o ka hale noho o J. Mānā, ua kūkulu ʻia aʻe la he lānai peʻa, a i laila kahi i hoʻomākaukau ʻia ai ʻo kekahi papa ʻaina. E hoʻuhene mau ana nō hoʻi ka pūʻali puhi ʻohe i kēlā a me kēia wā a hiki i ka wā ʻai, i ke kau pono o ka lā i ka piko a ma hope iho paha, me ka hoʻomau aku nō i ka hoʻoleʻaleʻa ʻana a pau ka papa ʻaina mua a ke Aliʻi, ʻo nā Kamāliʻi kāne nō ma kona a me ko kona kaikoʻeke wahi. ʻEkolu papa ʻāina i hoʻonoho ʻia ai a ua ʻai nā poʻe a pau i hiki aku a lawa pono, me ka hoʻouna ʻia o kahi mea-ʻai no kekahi poʻe i maopopo i hiki ʻole aku.

Ground-truth 6: Ma uka iki aku hoi o kēia māla, ma ka ʻaoʻao ma luna o ka hale noho o J. Mana, ua kūkulu ʻia aʻela he lānai peʻa, a i laila kahi i hoʻomākaukau ʻia ai o kekahi papa ʻāina. E hōʻuhene mau ana nō hoʻi ka pūʻali puhi ʻohe i kēlā a me kēia wā a hiki i ka wā ʻai, i ke kau pono o ka lā i ka piko a ma hope iho paha, me ka hoʻomau aku nō i ka hoʻoleʻaleʻa ʻana a pau ka papa ʻaina mua a ke aliʻi, o nā kamāliʻi kāne nō ma kōna a me kō kōna kaikoʻeke wahi. ʻEkolu papa ʻaina i hoʻonoho ʻia ai a ua ʻai nā poʻe a pau i hiki aku a lawa pono, me ka hoʻouna ʻia o kahi meaʻai no kekahi poʻe i maopopo i hiki ʻole aku.

Input 7: O na keiki puhiohe no hoi kekahi i kanu mau wahi mea kanu, a he mau mea ulu Hawaii wale no hoi ka lakou o ke ano papa kahuna. Mamuli o ka oluolu a me ka lokomaikai o kahi o lakou, ua loaa mai ia makou ka papa hoike a me na wehewehe ana o ka lakou.

Prediction 7: ʻO nā keiki puhi ʻohe nō hoʻi kekahi i ka_nū mau wahi mea kanu, a he mau mea ulu Hawaiʻi wale nō hoʻi kā lākou o ke ʻano papa kahuna. Ma muli o ka ʻoluʻolu a me ka lokomaikaʻi o k*ek*ahi o lākou, ua loaʻa mai iā mākou ka papa hōʻike a me nā wehewehe ʻana o kā lākou.

Ground-truth 7: ʻO nā keiki puhi ʻohe nō hoʻi kekahi i kanu mau wahi mea kanu, a he mau mea ulu Hawaiʻi wale nō hoʻi ka lākou o ke ʻano papa kahuna. Ma muli o ka ʻoluʻolu a me ka lokomaikaʻi o kekahi o lākou, ua loaʻa mai iā mākou ka papa hōʻike a me nā wehewehe ʻana o kā lākou.

Input 8: Mau Mea Kanu Hookalakupua

Prediction 8: Mau Mea Kanu Hoʻokalakupua.

Ground-truth 8: Mau Mea Kanu Hoʻokalakupua.

Input 9: o ke ano hookahuna oiaio maoli no. Eia iho no ia papa hoike:

Prediction 9: ʻo ke ʻano hoʻokahuna ʻoiaʻiʻo maoli nō. Eia iho nō ia papa hōʻike:

Ground-truth 9: ʻO ke ʻano hoʻokahuna ʻoiaʻiʻo maoli nō. Eia iho nō ia papa hōʻike:

Input 10: hala polapola. Ko puni e Kalani o ka lei e, leiia hoi o Halaomapuana, onaona i ka ihu, huihui ke hanu iho.

Prediction 10: hala polapola. Kō puni ē Ka_lani o ka lei ē, lei ʻia hoʻi ʻo Hālaomapuana, onaona i ka ihu, huʻihuʻi ke hanu iho.

Ground-truth 10: hala polapola. Kō puni e Kalani o ka lei ē, lei ʻia hoʻi ʻo Halaomāpuana, onaona i ka ihu, huʻihuʻi ke hanu iho.

First 10 sentences in Newspaper 2:

Input 1: O Kahahana, he alii kapu ia no o Oahu.
Prediction 1: ‘O Kahahana, he ali‘i kapu ia nō o O‘ahu.
Ground-truth 1: ‘O Kahahana, he ali‘i kapu ia no O‘ahu.

Input 2: O Kaionuilalahai ka makuahine, ka moopuna a Kalaniomaiheuila, ke kaikamahine a Kalanikahimakaialii, a laua o Kualu ke kaikuahine, a mua hoi o Kaulahea ka Moi o Maui.
Prediction 2: ‘O Ka ‘i‘onuilalahai ka makuahine, ka mo‘opuna a Kalani ‘ o ma‘i he uila, ke kaikamahine a Kalani kahi maka ‘ i ali‘i, ‘ā lāua ‘o Kū‘alu ke kaikuahine, a mua ho‘i o Ka‘ula hea, ka Mō‘ī o Maui.
Ground-truth 2: ‘O Ka‘ionuilalahai ka makuahine, ka mo‘opuna a Kalani‘ōmaiheuila, ke kaikamahine a Kalanikahimaka‘iali‘i, a lāua ‘o Kūalu ke kaikuahine, a mua ho‘i o Ka‘ulahea, ka Mō‘ī o Maui.

Input 3: A o ka makuakane, oia hoi o Elani, no ka ohana a Kupanihi, a o Keopuolani.
Prediction 3: A ‘o ka makua kāne, ‘o ia ho‘i ‘o ‘Elani, no ka ‘ohana a Kupanihi, a ‘o Keōpūolani.
Ground-truth 3: A ‘o ka makua kāne, ‘o ia ho‘i ‘o ‘Ēlani, no ka ‘ohana a Kūpānihi, a ‘o Keōpūolani.

Input 4: I ko Kahahana manawa kamalii, ua kii ia mai e Kahekili e lawe i Maui i keiki nana.
Prediction 4: I ko Kahahana manawa kamali‘i, ua ki‘i ‘ia mai e Kahekili e lawe i Maui i keiki nāna.
Ground-truth 4: I ko Kahahana manawa kamali‘i, ua ki‘i ‘ia mai e Kahekili e lawe i Maui i keiki nāna.

Input 5: Ua hanai kapu ia oia i Maui.
Prediction 5: Ua hānai kapu ‘ia ‘o ia i Maui.
Ground-truth 5: Ua hānai kapu ‘ia ‘o ia i Maui.

Input 6: A i kona lilo ana ae i kanaka makua, ua lilo oia i kanaka maikai, a ua nani hoi kona helehelena, a ua piipii maikai kona lauoho; a ua kapaia oia i kekahi wa, he piipii hahai moa.

Prediction 6: A i kona lilo ʻana aʻe i kanaka mākua, ua lilo ʻo ia i kanaka maikaʻi, a ua nani hoʻi kona helehelena, a ua piʻipiʻi maikaʻi kona lauoho; a ua kapa ʻia ʻo ia i kekahi wā, he piʻipiʻi hahai moa.

Ground-truth 6: A i kona lilo ʻana aʻe i kanaka makua, ua lilo ʻo ia i kanaka maikaʻi, a ua nani hoʻi kona helehelena, a ua piʻipiʻi maikaʻi kona lauoho; a ua kapa ʻia ʻo ia i kekahi wā, he piʻipiʻi hahai moa.

Input 7: I kona lilo ana i kanaka makua, hooipoipo aku la oia me na wahine kaukaualii, nolaila, ua ali ia ke kapu, a ua kapa hou ia ka inoa o Walia, a ua hoopauia ka inoa Ahi, Wela, Hahana.

Prediction 7: I kona lilo ʻana i kanaka mākua, hoʻoipoipo aku la ʻo ia me nā wāhine kaukaualiʻi, no laila, ua ʻali ʻia ke kapu, a ua kapa hou ʻia ka inoa ʻo Waliʻa, a ua hoʻopau ʻia ka ʻino a ʻAhi, Wela, Hahana.

Ground-truth 7: I kona lilo ʻana i kanaka makua, hoʻoipoipo akula ʻo ia me nā wāhine kaukaualiʻi, no laila, ua ʻali ʻia ke kapu, a ua kapa hou ʻia ka inoa ʻo Walia, a ua hoʻopau ʻia ka inoa Ahi, Wela, Hahana.

Input 8: Lawe ae la o Kahahana i wahine nana, o Kekuapoi ka inoa.

Prediction 8: Lawe aʻe la ʻo Kahahana i wahine nāna, ʻo Kekuapoʻi ka inoa.

Ground-truth 8: Lawe aʻela ʻo Kahahana i wahine nāna,ʻo Kekuapoʻi ka inoa.

Input 9: Ua olelo ia no hoi kela wahine, aohe ona lua iloko o ke aupuni Hawaii nei, a ua kaulana kona inoa mai Hawaii a Kauai, a ua lilo hoi i kaao, a ua kiekie hanohano kona kino; aole no hoi i ike ia kahi kina mai luna o ke poo a hiki i na kapuai wawae; he mau maka manu nunu kona i like me ko ka mohoea; a ua like hoi na helehelena o na hiohiona papalina me ka opuu rose i mohala maikai i ke kakahiaka; a ua nui na loli o kona mau hiohiona, i ke kakahiaka, i ke awakea, ahiahi a me ka po; o ka puo kelakela, oiai oia iloko o ia manawa, ua kupono ke ali ana o ke kapu.

Prediction 9: Ua ʻōlelo ʻia nō hoʻi kēlā wahine, ʻaʻohe ona lua i loko o ke aupuni Hawaiʻi nei, a ua kaulana kona inoa mai Hawaiʻi a Kauaʻi, a ua lilo hoʻi i kaʻao, a ua kiʻekiʻe hanohano kona kino; ʻaʻole nō hoʻi i ʻike ʻia kahi ʻkina mai luna o ke poʻo a hiki i nā kapuaʻi wāwae; he mau maka manu nūnū kona i like me ko ka moho e ʻā; a ua like hoʻi nā helehelena o nā hiʻohiʻona pāpālina me ka ʻōpuʻu rose i mohala maikaʻi i ke kakahiaka; a ua nui nā loli o kona mau hiʻohiʻona, i ke kakahiaka, i ke awakea, ahiahi a me ka pō; ʻo ka puʻo kelakela, ʻoiai ʻo ia i loko o ia manawa, ua kūpono ke ʻali ʻana o ke kapu.

Ground-truth 9: Ua ʻōlelo ʻia nō hoʻi kēlā wahine, ʻaʻohe ona lua i loko o ke aupuni Hawaiʻi nei, a ua kaulana kona inoa mai Hawaiʻi a Kauaʻi, a ua lilo hoʻi i kaʻao, a ua kiʻekiʻe hanohano kona kino; ʻaʻole nō hoʻi i ʻike ʻia kahi kīnā mai luna o ke poʻo a hiki i nā kapuaʻi wāwae; he mau maka manu nūnū kona i like me ko ka mohoea; a ua like hoʻi nā helehelena o nā hiʻohiʻona pāpālina me ka ʻōpuʻu loke i mōhala maikaʻi i ke kakahiaka; a ua nui nā loli o kona mau hiʻohiʻona, i ke kakahiaka, i ke awakea, ahiahi a me ka pō; ʻo ka puʻō kelakela, ʻoiai ʻo ia i loko o ia manawa, ua kūpono ke ʻali ʻana o ke kapu.

Input 10: Ua uluhua na'lii, na kahuna a me na makaainana o ke aupuni o Oahu i ko lakou Moi ia Kumahana, i ke keiki a Peleioholani.

Prediction 10: Ua uluhua nā ‘’*a*liʻi, nā kāhuna a me nā makaʻāinana o ke aupuni o Oʻahu i ko lākou Mōʻī, iā Kūmahana, i ke keiki a Peleiōhōlani.

Ground-truth 10: Ua uluhua nā aliʻi, nā kāhuna a me nā makaʻāinana o ke aupuni o Oʻahu i ko lākou Mōʻī, iā Kūmahana, i ke keiki a Peleiōhōlani.