Computational Methods for Higher Accuracy Nanopore Sequencing

by

Jordi Joaquim Silvestre-Ryan

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Joint Doctor of Philosophy
with University of California, San Francisco

in

Bioengineering

and the Designated Emphasis

in

Computational and Genomic Biology

in the

Graduate Division

of the

University of California, Berkeley

Fall 2022

Computational Methods for Higher Accuracy Nanopore Sequencing

Abstract

Computational Methods for Higher Accuracy Nanopore Sequencing

by

Jordi Joaquim Silvestre-Ryan

Doctor of Philosophy in Bioengineering

and the Designated Emphasis in

Computational and Genomic Biology

University of California, Berkeley

Professor Ian Holmes, Chair

Nanopore sequencing is a versatile technology that can generate long, single-molecule reads on a portable device. This presents strong advantages in areas such as metagenomics and de novo genome assembly. These advantages have traditionally been tempered by an error rate higher than other sequencing technologies, though advancements in both sequencing chemistry and basecalling models have yielded steady increases in sequencing accuracy. This work presents multiple computational techniques for further reducing this error rate, primarily by combining information from multiple noisy reads into a single, higher accuracy consensus sequence. We present algorithms for using the probabilistic output of existing basecallers to find the consensus of two (Chapter 2) or more (Chapter 3) reads. We also introduce a neural network polisher for multi-read consensus at higher read depths (Chapter 4). Finally we explore the application of policy gradients, a technique developed for reinforcement learning, to train nanopore basecallers (Chapter 5). These methods are implemented in a variety of software tools, all of which are freely available on GitHub (`https://github.com/jordisr/`). Our software PoreOver is designed to work with the output of multiple basecallers and implements two main consensus algorithms. We also present PoreOverNet, a simple standalone basecaller. In addition to the standard maximum likelihood loss generally used to train most basecallers, PoreOverNet also implements a multi-objective loss designed to reduce the expected number of errors through a policy gradient approach. Finally, Semapore is a neural network tool for consensus and polishing of genome assemblies.

For my parents

Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I need to thank my advisor, Ian Holmes. Without his support and belief in me, I would likely not have completed a PhD. I'd also like to acknowledge the other members of my thesis committee, Michael Keiser and Nir Yosef, for their encouragement and thoughtful comments.

I would also like to thank Daniel Fletcher. I've felt welcome in his lab since I first returned to Berkeley, and am so grateful for the many scientific and social interactions I've had with him and the members of his lab.

I would additionally like to acknowledge my graduate schools peers: the 2016 cohorts of the Bioengineering and Computational Biology PhD programs. Activities like snowy Tahoe cabin trips, scientifically rigorous margarita tastings, and filming sketch videos for the department retreat have all provided very welcome distractions.

My path to this degree has not been particularly direct, but ultimately I've learned and grown at each step of my scientific journey. As such, I'd like to thank some of the many scientists who have guided and mentored me along the way: Michael Syvanen, Jhih-Wei Chu, Xavier Salvatella, Santiago Esteban-Martín, and Raj Bhatnagar.

Last but not least, I owe a debt of gratitude to all of my friends and family, near and far, whose support over the years has truly been invaluable. My partner Amanda has been a constant source of humor and joy, and I'm so glad we were able to support each other through the ups and downs of graduate school. Finally, my parents, Maureen and Joaquim, have provided unwavering support my entire life, and I'm very grateful for all the ways they've nurtured my creativity and curiosity from a young age.

Chapter 1

Introduction

1.1 DNA sequencing

Nucleic acids are the language of life. An organism's genome serves as a DNA blueprint: it encodes the sequences for all the cell's proteins and RNA, as well as the complex instructions for building and regulating this machinery. With the technology of DNA sequencing we can read back this language of four nucleic acid bases: adenine (A), cytosine (C), guanine (G), and thymine (T). The sequencing of genes and entire genomes allows us to better understand ourselves, other organisms, and the environment we live in.

At the onset of the Human Genome Project in 1990, sequencing was a laborious process, and as such the project involved a massive network of collaborating genomics centers across the world. The state-of-the-art technique at the time was Sanger sequencing, which uses gel electrophoresis and fluorescently labeled dideoxynucleotides to synthesize and read DNA in fragments of roughly 1000 base pairs (bp) at a time [22]. With increasing improvement and automation, the cost per sequenced base decreased multiple orders of magnitude over the course of the human genome project [11]. Since then, DNA sequencing technology has continued to progress by leaps and bounds and has become even more accessible and affordable.

The next generation of sequencing, pioneered by Illumina and Solexa, uses growing clusters of identical DNA fragments anchored to a flow cell [41]. These microscopic clusters undergo several cycles where clusters are extended by a nucleotide and then imaged fluorescently. Under this method, DNA is fragmented into short segments that can be read in a highly accurate and high-throughput method. While the per base accuracy of short read sequencing is very high (>99.9% accurate), the reads it generates are 300 bp or shorter, which poses a computational challenge for some downstream tasks. For example, de novo genome assembly uses overlaps between reads to piece together contiguous regions, which is more difficult to do with short reads and especially for repetitive regions of the genome.

While short-read Illumina sequencing is still the de facto sequencing technology, there have been exciting developments over the past decade in long-read sequencing techniques that can generate reads orders of magnitude longer, thus avoiding some of these issues. In particular, commercial long-read sequencing has been spearheaded by Oxford Nanopore Technologies (ONT) and Pacific Biosciences (PacBio).

Sequencing on the ONT platform can generate reads tens or even hundreds of kilobases long, and can be done on the portable MinION device [27]. Thanks to long reads, the human genome has finally been completed down to the repetitive telomere regions [44]. The portability of the MinION device has enabled environmental monitoring [50] and pandemic surveillance [18] far from traditional laboratories, and allowed sequencing to be done in locations as remote as Antarctica [28] and the International Space Station [6]. Additionally, nanopore sequencing is a single molecule technique, and can also be used to assay methylation (and other base modifications) and directly sequence RNA without reverse transcription back to DNA [51][52].

Despite these relatively unique advantages, nanopore sequencing has historically been limited by its error rate, which is higher than short-read techniques [53]. This error rate has been brought down significantly through advances in both the sequencing chemistry as well as the algorithms and machine learning models used to analyze the raw sequencing data. This thesis presents several methods for addressing this latter set of computational challenges. This introduction provides an overview of nanopore sequencing and basecalling, the process of mapping the raw electrical signal generated during sequencing back to a sequence of nucleic acid bases. To provide the necessary background we will first introduce in more depth how nanopore sequencing works and how its basecalling has evolved over time. As modern basecalling is exclusively done with neural networks, this introduction also includes a brief primer on deep learning.

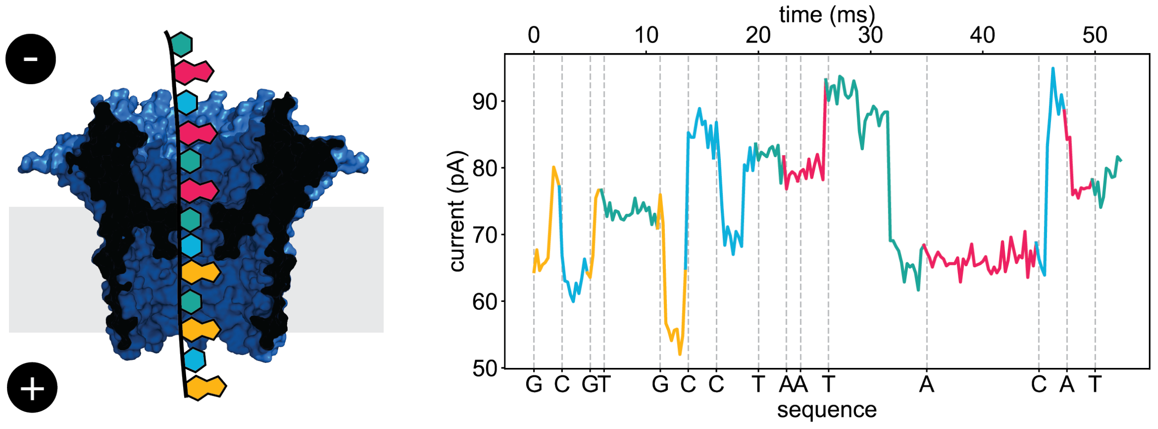


Figure 1.1: Overview of nanopore sequencing.

As a single strand of DNA passes through the pore it disturbs the electrical current in a complex, sequence-dependent manner. Basecalling uses machine learning to infer the sequence of bases that went through the pore. The structure of the *E. coli* protein CsgG (PDB 4Q79) is used in this diagram and was one of the early pore proteins used by ONT [5].

1.2 Sequencing with nanopores

The general concept of nanopore sequencing is to run a voltage across a membrane, with a small opening or pore just large enough for single-stranded DNA to pass through. The flow of ions through this pore generates an electrical current which can be measured. As the DNA backbone is negatively charged, a single-stranded DNA molecule will get pulled through the opening to the positively charged side of the membrane. As the DNA electrophoretically moves through the pore, it blocks the flow of ions and leads to a reduction in the measured current levels. Crucially, this perturbation in the current level is sequence-specific and dependent on the nucleotides in the pore. Using machine learning techniques, the DNA sequence can be inferred from the electrical current in the step of basecalling. Figure 1.1 gives an overview of this process and the data that is generated. While the idea of using nanopores to sense or sequence DNA was conceived over three decades ago [13], the technology was not widely available until the early access release of the MinION sequencing platform by Oxford Nanopore Technologies in 2014 [27].

Through these intervening years, different types of pores have been explored for this purpose, from mutated biological pore proteins to engineered solid-state graphite pores. Much initial development of nanopore sequencing focused on the α-hemolysin protein from *Staphylococcus aureus* and established that different DNA sequences led to measurable differences in the current [14]. The MspA protein from *Mycobacterium smegmatis* was later used, as it allowed for a clearer separation in the signal of each nucleotide [15]. While ONT at one point used a derivative of the CsgG pore protein from *Escherichia coli* [5], the current pore identity is not known.

Besides the pore protein itself, effective nanopore sequencing also requires another protein to help the DNA through the pore. With electrophoresis alone, the DNA would actually move through the pore too fast to effectively resolve individual bases [15], so polymerases or helicases are used to help unwind double-stranded DNA and ratchet it through the pore at a slower speed. While Phi29 DNA polymerase was used successfully in earlier academic studies [35], the identity of the "motor" protein that ONT currently uses is not publicly known.

Since the first release of the MinION, ONT has upgraded the sequencing chemistry several times, introducing newer pores and sequencing kits with higher accuracy [53]. In parallel to this, there have also been improvements to the basecalling software, which is covered in more detail in Section 1.4. One powerful way of increasing the accuracy using error-prone reads is to sequence the same DNA sequence multiple times, and then combine this information to generate a higher accuracy consensus sequence. This has been the subject of work by ONT, who have developed protocols to read both strands of a single DNA molecule using special hairpin adapters, which allow the complementary strand to follow after the first strand goes through the pore. ONT has commercialized multiple variations of this technique, which was known initially as 2D sequencing, then $1D^2$, and more recently as Duplex sequencing. The current signatures of both strands can be combined for a higher accuracy sequence. One such method is presented in Chapter 2, which has since been adopted into official ONT software.

Going beyond two reads, one would ideally read the same sequence many times to generate the highest accuracy consensus sequence. By reversing the polarity of the voltage across the membrane, it seems possible to shuttle the molecule back and forth read the same molecule multiple times. This is something that ONT have shown a proof-of-concept of[1], but is not yet commercially available on their platform. Nevertheless, while single-molecule re-reading is not yet possible, there are other ways of generating multiple copies of a sequence of interest. One such technique [64] uses rolling circular amplification to create a molecule with several copies of the same target sequence. Alternatively, unique molecular barcodes (UMIs) can be used to tag amplicons such that reads can be clustered by amplicon and used to generate high accuracy sequences [30]. This clustered amplicon data is the focus of Chapter 3. Lastly, in whole genome sequencing, multiple reads may overlap at the same position of the genome or assembly. There are a number of techniques to do consensus and polishing on this type of data, one of which is presented in Chapter 4. After this brief introduction to the science behind nanopore sequencing, we will move toward an overview of basecalling, but first covering the basics of the neural networks which underlie all modern basecallers.

1.3 A neural network primer

Though first described in the 1960's, neural networks have risen to widespread prominence in the past two decades. Early successes in computer vision [36], speech recognition, and machine translation have shown the power of neural network-based approaches and contributed to their ubiquity. Biology, like many other disciplines, has been radically affected by the rise of deep learning and has seen adoption of neural networks for tasks such as analyzing single-cell transcriptomics [39] and protein structure prediction [29]. Provided with sufficient data, neural networks have been shown to learn generalizable patterns and make predictions with accuracies exceeding earlier approaches without deep learning. Nanopore sequencing has been no exception, and all but the earliest generation of nanopore basecallers have been built from neural networks. This section thus provides a concise overview of the developments in deep learning that are relevant for nanopore basecalling. For a comprehensive introduction to the subject of deep learning, see Goodfellow [19]. Following this introduction to more general neural network concepts, the Section 1.4 provides a timeline and overview of nanopore basecallers.

Feed-forward neural networks

At a high level, a neural network takes an input x and transforms it to an output y. This output could be a scalar, as in a regression task that seeks to predict a single value, or it could be vector, for instance of class probabilities in a classification task. The simplest neural network (Figure 1.2A) consists of an input layer (x), a single hidden layer (h), and an output layer (y), and is known as a feed-forward network or alternatively a multilayer perceptron. Each layer is associated with a set of variables, vectors or matrices which capture the state of a neural network at a given time. The parameters of the neural network are the weight matrices (W_{xh}, W_{hy}) associated with each layer, and are learned during training.

[1]Clive Brown, London Calling 2021, https://www.youtube.com/watch?v=AlAnmvbNwfY

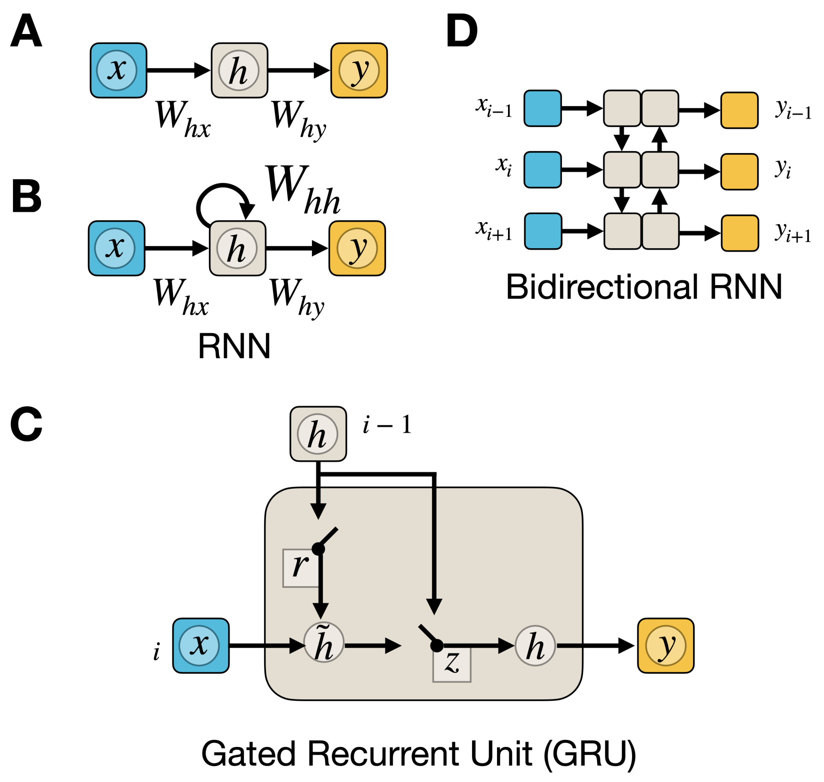


Figure 1.2: Neural networks for modeling sequences.

(A) Basic feed-forward network with a single hidden layer. (B) Recurrent neural network (RNN), used to model variable-length sequences. (C) Gated recurrent units (GRU) can learn to selectively propagate hidden states, allowing them capture longer-range information better than a standard RNN. (D) Bidirectional networks combine two layers to propagate information both forward and backwards in a sequence.

The product of the weight matrix and previous layer is passed through a nonlinear function f, which is known as an *activation function.*

$$h = f(W_{xh}x)$$
$$y = f(W_{hy}h)$$

Common choices for the activation function are hyperbolic tangent (tanh) or other sigmoidal or step functions.

Larger networks can be created by composing multiple hidden layers, alternating these linear matrix multiplications and nonlinear activation functions. This inclusion of nonlinearity is crucial for the neural network's power. Without these activation functions, a multi-layer network would reduce to a single linear transformation, thus greatly reducing its expressivity.

Training the network requires a collection of labeled (x, y^*) pairs, where x is an example input and y^* is the true, known output. During training, a forward pass of the network yields the predicted output y, which is compared to the true value y^*. A *loss function* quantifies how far the predicted and true values are, and is the objective to be minimized during training. Common loss functions include the root mean-square error for regression or negative log-likelihood for classification tasks. The minimization of this objective function is usually done with gradient descent, which calculates the gradient of the loss with respect to the parameters (the weight matrices), and takes a small step in the direction opposite the gradient (downhill). This is done stochastically, by iterating over the data in small subsets (*batches*) and updating the weights according to the gradient. In practice, training can converge faster using advanced minimization schemes, such as Adam [31], which also make use of higher order derivatives.

Recurrent neural networks

While feed-forward networks have inputs and outputs of a fixed length, there are many machine learning tasks that need to handle sequences of variable length. These could be sequences of words for machine translation, audio waveform levels for speech recognition, or the electrical current time series measured during nanopore sequencing (as in Figure 1.1). Consider a sequence $\boldsymbol{x}$ of length T:

$$\boldsymbol{x} = (x_1, x_2, x_3, ..., x_T)$$

The recurrent neural network (RNN) was developed to model sequences by having a hidden layer that depends on the hidden layer of the previous input (Figure 1.2B).

$$h_i = f(W_{xh}x_i + W_{hh}h_{i-1})$$
$$y_i = f(W_{hy}h_i)$$

As the simple RNN only captures time dependence in one direction, it is common to take two RNN layers running in opposite directions over a sequence and concatenate their outputs, thus producing a bidirectional RNN (Figure 1.2D).

While the simple RNN is in theory capable of propagating information across long sequences, in practice this is quite difficult, with gradients becoming too small (vanishing) or too large (exploding) [49]. One solution is to replace each element of a layer with a gated cell that can selectively 'remember' and 'forget'. This circumvents the vanishing gradient problem and has seen much success. The Gated Recurrent Unit (GRU, [9]) and Long Short-Term Memory (LSTM, [23]) are two common approaches for doing this. The LSTM was developed much earlier, though the GRU is slightly simpler with similar performance [10] and is shown in Figure 1.2C.

The GRU cell can be thought of as having two gates. The update gate (z) determines whether the current hidden state should update or just copy the previous hidden state, which is done by taking a linear combination of the previous hidden state and the proposed hidden state ($\tilde{h}$):

$$h_i = (1 - z_i)h_{i-1} + z_i\tilde{h}_i$$

The proposed hidden state is calculated similarly to a standard RNN but with the addition of a second gate, the reset gate (r). When the reset gate is 0, the GRU only has information from the input and is treated as the first element of a normal RNN and is effectively reset ($\odot$ denotes the elementwise product).

$$\tilde{h}_i = \tanh\left(W_{xh}x + W_{hh}(r_i \odot h_{i-1})\right)$$

The activations of the gates are calculated at each time step using their own set of weight matrices: $W_{xz}, W_{hz}, W_{xr}, W_{hr}$.

$$z_i = \sigma(W_{xz}x + W_{hz}h_{i-1})$$

$$r_i = \sigma(W_{xr}x + W_{hr}h_{i-1})$$

Connectionist temporal classification

As described previously, the loss function quantifies the accuracy of a neural network's prediction, and is the objective to be minimized during training. This section provides an overview of one such loss function, connectionist temporal classification (CTC, [21]), which was originally developed for speech recognition but has been adopted for nanopore basecalling.

In speech recognition, a key task is taking an input sequence of an audio waveform and segmenting it into words or individual phonemes. As RNNs generate an output for each input, the network is set up to output normalized probabilities over all possible labels (four in the case of DNA sequencing) plus an additional blank or gap character. This gap character allows the network to output a sequence of labels that is much shorter than the input sequence. For example, with nanopore sequencing, if the current is sampled at 4000 Hz and the DNA moves through the pore on average at 450 bp/s, you would expect 9 current measurements per base, which would correspond to an output of roughly one label and 9 gap characters.

To illustrate what the output of a CTC RNN might look like, let's consider a toy problem where we have an alphabet with just two characters (A and B) instead of the four bases.

Given an input of length 4, suppose we get the following output from our neural network.

$$\begin{array}{cc} t = & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} A \\ B \\ - \end{array} & \begin{bmatrix} 0.6 & 0.2 & 0.1 & 0.3 \\ 0.3 & 0.1 & 0.2 & 0.6 \\ 0.1 & 0.7 & 0.7 & 0.1 \end{bmatrix} \end{array}$$

Each four character permutation of this expanded alphabet, e.g. `A-B-`, corresponds to a path through the output. The probability of a path is easily obtained by multiplying probabilities of that character in each column

$$P(\texttt{A-B-}) = (0.6)(0.7)(0.2)(0.1) = 0.0084$$

The probability of a label is the sum of all paths that yield that label once gaps are removed, essentially marginalizing over all the possible alignments between the input and the output labeling.

$$\begin{aligned} P(\text{AB}) &= P(\texttt{AB--}) + P(\texttt{A-B-}) + P(\texttt{A--B}) + P(\texttt{-AB-}) + P(\texttt{-A-B}) + P(\texttt{--AB}) \\ &= 0.202 \end{aligned}$$

Naively training a classifier to do this task would require training data with an explicitly specified alignment, whereas using CTC allows for multiple possible alignments. Training is done by minimizing the negative log-likelihood of the true label sequence. Rather than explicitly enumerating paths as in the above example, the label probability can be calculated efficiently with dynamic programming (see Section 2.4).

For inference, we wish to find the best label after having generated these probabilities. While the best label can be found with a slow prefix search, in practice either a greedy approach like beam search [20] or just taking the best path (in this case the best character in each column), which corresponds to the Viterbi decoding, is used.

1.4 A brief history of nanopore basecalling

In conjunction with updates to the pore chemistry, much of the improvement in sequencing accuracy has come from better methods for basecalling [53]. In several cases, these developments came originally from the academic community, to be later adopted in official software by Oxford Nanopore Technologies. This section attempts to give a brief overview of the evolution of nanopore basecalling methods.

The earliest generation of basecallers did not work with the raw levels of current signal directly but rather a preprocessed version that had been segmented into discrete "events". Each event was associated with a duration (how many signal measurements it covered), a mean current level, and the standard deviation of the current. The initial ONT basecaller (Metrichor) as well as the first open-source basecaller (nanocall [12]) both used hidden Markov models (HMMs) to classify these events using k-mers, which represented the short sequence of bases centered in the pore at a given time. Nanocall's HMM was designed to work with the pore model developed by ONT, a description of the mean/std current for each of 4096 6-mers. Nanocall had a state for each of these 6-mers, and sought to model the sequence of events as a path through these 6-mer states. In theory, sequential k-mers should overlap for $k-1$ bases, but in cases where procession through the pore happened too quickly, there may be skipped k-mers in the events, something nanocall addressed by adding in additional transitions between k-mers that were near each other in sequence.

As deep learning grew in popularity and recurrent neural networks got applied to a wider variety of sequence classification tasks, in hindsight it seems inevitable that neural networks would make their way to nanopore basecalling. An early RNN basecaller was DeepNano [3], which used bidirectional GRUs to basecall from segmented events. For each event, the neural network output a probability distribution over 4 bases plus a blank character, to account for repeated events. Around this time ONT also switched to a deep learning approach with neural network basecallers Nanonet and Albacore.

Another important transition came in the shift away from preprocessed events and k-mer calling to basecalling directly from the raw signal. The basecaller Chiron [61] applied the CTC loss from speech recognition to yield competitive accuracies, particularly for genome assembly and consensus applications [65]. Other community basecallers like DeepNano-blitz [4] adopted CTC; and ONT released the Guppy basecaller, which for the last several versions has run on different modifications of the CTC approach.

In addition to ONT's main basecaller used in production and shipped with the sequencing platform, it has also maintained research basecallers to test out new models and strategies, which, when successful, are then adopted into the production software. Among these are the "flip-flop" CTC model developed in the basecaller Flappie[2], and run-length encoding in the basecaller Runnie[3].

The "flip-flop" modification of CTC was developed as an alternative to better model homopolymers. Under this model, there are no gap characters but instead an expanded alphabet of 8 bases: 4 in the "flip" state and 4 in the "flop" state. Repeated characters emitted from the same flip or flop state are merged, and transitions between flip and flop states are only allowed for the same base. Thus, the only way to emit a homopolymer is by alternating between flip and flop states (see section 2.4). Another approach for homopolymers was to explicitly model the length of the homopolymer in addition to the base identity. This would generate the run-length encoding of the basecalled sequence and was tested with the basecaller Runnie, which would output both a base and the parameters of a discrete Weibull distribution governing the homopolymer length.

Among the most successful of these research basecallers has been Bonito. Bonito was

[2]`https://github.com/nanoporetech/flappie/`

[3]`https://github.com/nanoporetech/flappie/blob/master/RUNNIE.md`

originally inspired by the convolutional architecture of Nvidia's QuartzNet [33] and implemented a standard CTC model (in contrast with the flip-flop model used by Guppy). The latest iteration uses a hybrid CTC/conditional random field (CRF), where the output can be interpreted as weights in a linear-chain CRF [34]. These CRF models break the independence assumptions of the standard CTC model, but this increased expressivity has yielded clear accuracy gains for nanopore basecalling [48]. Despite many advances in architectures for other sequence modeling tasks, recurrent networks still perform well in nanopore basecalling, and the choice of loss function whether CTC or hybrid CTC/CRF seems more important than the architecture. Interestingly, the popular transformer architecture [63] seems to perform worse than RNNs for basecalling [48].

Chapter 2

Pair consensus decoding improves accuracy of neural network basecallers for nanopore sequencing

2.1 Abstract

We develop a general computational approach for improving the accuracy of basecalling with Oxford Nanopore's 1D^2 and related sequencing protocols. Our software PoreOver (`https://github.com/jordisr/poreover`) finds the consensus of two neural networks by aligning their probability profiles, and is compatible with multiple nanopore basecallers. When applied to the recently-released Bonito basecaller, our method reduces the median sequencing error by more than half.

2.2 Main text

Nanopore sequencers, such as the MinION and related devices from Oxford Nanopore Technologies (ONT), allow for direct readout of individual DNA molecules [13]. However, the higher error rate of nanopore sequencing compared to other methods has limited its application in situations where deep coverage is unavailable, such as detection of rare variants or characterization of highly polymorphic samples. In principle, 2X coverage is available even for single duplexes, using ONT's 1D^2 protocol or related methods which sequence both strands of the duplex consecutively. In the 1D^2 protocol, special DNA adapters are used such that after the template DNA strand passes through the pore, its complementary strand very often follows. Combining the readout of both strands should improve accuracy; however, most neural network basecaller architectures are designed to operate on single strands. Here we present a general method for adapting existing basecallers to take advantage of the extra information in paired 1D^2 reads.

Nanopore sequencing works by threading a single strand of DNA through a protein nanopore embedded in a synthetic membrane. The DNA bases block the pore, perturbing the ionic current flowing through. The current can be measured, and the original sequence of nucleotides recovered computationally. This latter *basecalling* step makes heavy use of machine learning techniques and, increasingly, of neural networks.

Early neural network basecallers (such as DeepNano[3], BasecRAWller[60], and certain ONT-developed basecallers) relied on a preprocessing step that segmented the current measurements into discrete events, corresponding to individual nucleotides passing through the pore. This aspect of basecalling shares similarities with speech recognition, where an audio time series must be segmented and then labeled with phonemes. Inspired by this similarity, later basecallers used Connectionist Temporal Classification (CTC), a method developed for speech recognition, which trains neural networks to do segmenting and classification simultaneously [21]. The community basecaller Chiron [61] successfully applied CTC to nanopore

A version of this chapter was originally published as: Silvestre-Ryan, J., & Holmes, I. (2021). Pair consensus decoding improves accuracy of neural network basecallers for nanopore sequencing. Genome Biology, 22(1), 38.

basecalling [65], while ONT incorporated CTC-style models into both production and research basecallers.

A CTC-trained neural network outputs a probability profile (Figure 2.1A) defining a distribution $P(\ell|y)$ over possible basecalled sequences ℓ given the read y. By analogy to hidden Markov models, the task of finding the modal sequence of this distribution is termed "decoding". While perfectly optimal decoding requires an intractably exhaustive search over sequences, heuristic algorithms (such as beam search or Viterbi search) can in practice be used to find reasonably good solutions.

The related task of "consensus decoding" arises when multiple reads $\{y_n\}$ are derived from the same underlying sequence ℓ, as is the case for $1D^2$. Basecalling then yields multiple profiles $P(\ell|y_n)$. Our task is to find the single sequence that maximizes $P(\ell|\{y_n\})$; under a flat prior $P(\ell)$ and the assumption that the reads are independent, this will be the sequence that maximizes the product $\prod_n P(\ell|y_n)$, motivating the reframing of this problem as an exercise in profile-profile alignment [58].

To this end we have developed a beam search decoding algorithm for the pair decoding of two reads, making use of a constrained dynamic programming heuristic to speed calculations by focusing on areas of each read which are likely to represent the same sequence (full details provided in Section 2.4). We introduce our basecalling software PoreOver, which implements these decoding algorithms and includes a basic recurrent neural network basecaller (PoreOverNet) for demonstration purposes.

DNA flows through the pore at an average of 450 bases/second; the electrical signal is recorded at 4000 Hz, yielding 9 measurements/base on average. Thus, if a read represents T bases, aligning two basecalled reads will take $\sim T^2$ steps, but aligning the raw signal measurements will take $\sim (9T)^2$ steps—an 81-fold increase compared to aligning basecalled sequences. To accelerate calculations we constrain our heuristic search to an "alignment envelope" containing the timepoints where the reads are most likely to align [24].

This envelope is estimated by doing a preliminary Viterbi decoding step on each read individually, then aligning the two sequences so obtained. This is faster than beam search, with similar performance (see Section 2.4), and explicitly maps each nucleotide to some range of timepoints. The two decoded sequences are then aligned globally, generating a nucleotide-level mapping between the reads, and (by extension) between the underlying time series. With some additional padding, this guide alignment defines the envelope for our banded 2D beam search (Figure 2.1B).

As nanopore reads can vary in length over orders of magnitude, a naive Needleman-Wunsch alignment may involve creating infeasibly large dynamic programming matrices. As a workaround, we use a modified Needleman-Wunsch with a fixed diagonal band. This appears to be sufficient for subsequent pair decoding, though exploiting recent advances in efficient pairwise alignment algorithms (such as [40]), may yield further improvements in accuracy and speed.

We tested our pair decoding algorithm on a sample of 5,000 R9.4 *E. coli* $1D^2$ read pairs (Oxford Nanopore Technologies, personal communication), comprising 10,000 reads in total.

Reads were run through a forward pass of our PoreOverNet basecaller to generate softmax probabilities, which were used for subsequent pair decoding.

After pair decoding, reads were aligned to the reference *E. coli* genome with Minimap [37] and the read accuracy calculated as (number of matches)/(length of alignment). We find that our banded 2D beam search improves the median accuracy from 87.6% for single reads to 93.2% for $1D^2$ read pairs (Figure 2.2), nearly halving the error rate of our PoreOverNet basecaller.

Our software can readily be adapted to work with the output of other neural network basecallers. Application to the recent DeepNano-blitz [4] showed a similar gain in accuracy from consensus decoding. We also applied our algorithm to the ONT basecaller Bonito[45], a research basecaller inspired by recent successes of purely convolutional neural networks in speech recognition, and compared results with Guppy, an earlier ONT basecaller which can make use of $1D^2$. Our consensus method lifts Bonito's median accuracy from 94.7% to 98.1%, better than halving the median error rate for single read basecalling and surpassing the consensus accuracy of Guppy's $1D^2$ method (Figure 2.2). Unlike Guppy, our code is open source; further, it is modular in design, making it straightforwardly modifiable and re-usable for other basecallers. We thus envision the PoreOver as a consensus decoding tool to be used in concert with a state-of-the-art CTC basecaller such as Bonito.

Generalizing beyond a pair of reads, consensus approaches are relevant to *polishing*, the task of refining a draft genome assembly by realigning reads to the draft. There are several approaches to polishing via multi-read consensus: some analyze the raw current signal using a hidden Markov Model [38] or dynamic time warping [7], while others analyze the basecalled sequence using neural networks [56, 46]. To our knowledge, none of the neural network methods explicitly use the intermediate basecaller probabilities (instead relying on previously basecalled sequence), while the methods that do use the raw signal do not use neural networks. The pairwise dynamic programming approach we describe could be extended to multiple reads, although the curse of dimensionality (a full dynamic programming alignment of N reads takes $\mathcal{O}(T^N)$ steps) would necessitate additional heuristics to narrow down the search space. These could include generalizing alignment envelopes to multiple sequences, or performing a stochastic search. With such heuristics, it should be possible to implement an algorithm to exploit the basecaller probabilities for general, multi-read consensus [58].

2.3 Declarations

Availability of data and materials

Our software PoreOver is available at `https://github.com/jordisr/poreover` under an MIT license. The *E. coli* 1D^2 reads used to test our pair decoding algorithm were generated by Oxford Nanopore Technologies and are available at `https://figshare.com/articles/dataset/E_coli_1D2_nanopore_sequencing_reads/13415867/1`.

Competing interests

The authors received research funding (IHH) and travel reimbursement (JSR) from Oxford Nanopore Technologies.

Funding

The authors were supported by NIH/NCI grant CA220441, NIH/NHGRI training grant T32 HG000047, and by a research gift from Oxford Nanopore Technologies.

Authors' contributions

JSR developed the software and conducted the benchmark analysis. JSR and IHH wrote the manuscript.

Acknowledgements

We thank Tim Massingham and Marcus Stoiber (Oxford Nanopore Technologies) for helpful discussion, and the anonymous reviewers for their feedback and suggestions. This work used the computational cluster provided by the Berkeley Research Computing program.

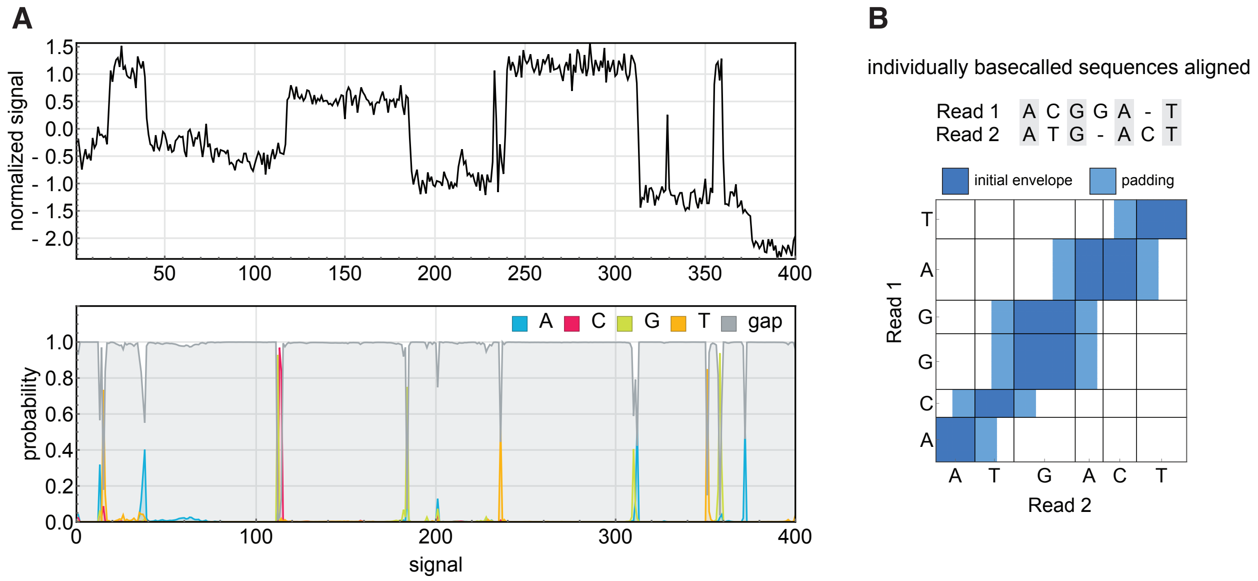


Figure 2.1: Nanopore basecalling maps signal to sequence.

(A) To basecall a single read, the time series of current signal is fed into a neural network basecaller which outputs for each measurement the probabilities of each base plus a blank gap character. This probability profile is then decoded to find the most likely basecalled sequence.
(B) To constrain our pair decoding algorithm, each read was basecalled individually and the alignment of the resulting sequences was used to define a region in signal space that banded our 2D beam search.

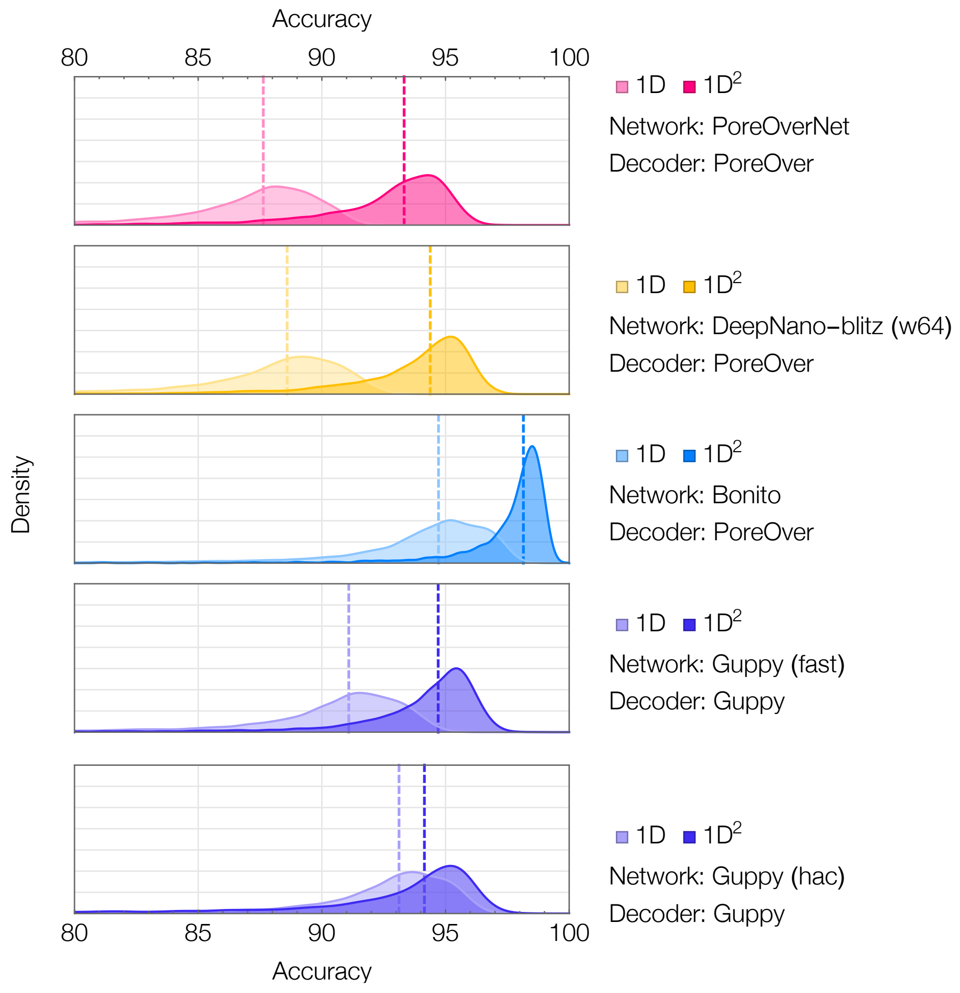


Figure 2.2: Consensus decoding improves sequencing accuracy.

Reads were run through PoreOverNet (magenta), the community basecaller DeepNano-blitz (yellow), and ONT's Bonito basecaller (blue) to generate softmax probabilities, which were then decoded using our algorithms. Guppy accuracies (in violet) were generated entirely from running the Guppy basecaller and its $1D^2$ basecalling mode without any additional decoding. The Guppy basecaller has the option of two neural network architectures using either smaller (fast) or larger (high accuracy, hac) recurrent layer sizes. DeepNano-blitz was run with its width64 network. The median accuracy is represented by a dashed line.

2.4 Supplementary Information

Basecalling with connectionist temporal classification

Under CTC, the output of the neural network defines a probability distribution over possible labelings of the input, a "labeling" in this case representing the DNA sequence that passed through the pore. CTC uses a differentiable loss function calculated with dynamic programming to calculate the probability of a given labeling. Using gradient descent, the network is trained to maximize the probability of the correct sequence.

The final softmax layer of the neural network outputs y, a $5 \times T$ matrix that specifies the probability of emitting each base plus a gap character at each step of the input. Let $y(t, c)$ be the probability of the character $c \in \mathcal{G}$ at time t, with $\mathcal{G} = \mathcal{L} \cup \{\epsilon\}$, where characters in $\mathcal{L} = \{A, C, G, T\}$ represent bases passing through the pore and ϵ is a gap or blank character representing no change in the pore. The neural network output can thus be interpreted as a linear hidden Markov model [58], with the softmax probabilities corresponding to emission probabilities in this HMM. The probability of a given path through this profile $\pi \in \mathcal{G}^T$ is just the product of the individual probabilities

$$P(\pi|y) = \prod_{t=1}^{T} y(\pi_t, t)$$

Under this model, sequences longer than T have zero probability.

Each gapped path π can be mapped to an ungapped label sequence ℓ by a function $\mathbf{B} : \mathcal{G}^* \to \mathcal{L}^*$, which simply removes the gap characters. This is a simplifed version of the path-to-label mapping used by the canonical CTC model [21]; unlike the original, our version does not merge repeated label characters (so the path `A-CCG--T` would result in the label `ACCGT` rather than `ACGT`). For a given label sequence ℓ, we want to find the probability that ℓ was emitted by y, $P(\ell|y)$. The probability of a label sequence is the sum of probabilities of all paths consistent with it, essentially marginalizing over all possible positions of gaps:

$$P(\ell|y) = \sum_{\pi:\mathbf{B}(\pi)=\ell} P(\pi|y)$$

This probability can be efficiently computed by dynamic programming with a Forward algorithm [16]. We define the Forward probability of a label as $\alpha(t, s)$, the probability that the first s characters of ℓ having been emitted by position t of the underlying HMM. The core recursion is,

$$\alpha(t, s) = y(t, \ell_s)\alpha(t-1, s-1) + y(t, \epsilon)\alpha(t-1, s)$$

terminated by the base case $\alpha(0, 0) = 1$. From this matrix we can easily read out the probability of the full sequence, $P(\ell|y) = \alpha(T, |\ell|)$.

Decoding the basecaller output

For basecalling we want to find the best label, $\hat{\ell}$,

$$\hat{\ell} = \text{argmax}_{\ell} P(\ell|y)$$

Borrowing HMM terminology, this task is referred to as decoding. While there is not an efficient general algorithm for this optimization, various heuristic search algorithms can be used to find high probability sequences. Here we focus on two approximate methods, finding the (1) Viterbi best path, and (2) a beam search.

While finding the best label is intractable, a simpler approach is to instead find the single best path:

$$\hat{\pi} = \text{argmax}_{\pi} P(\pi|y)$$

Thus, the Viterbi solution is $\ell_{\text{Viterbi}} = \mathbf{B}(\hat{\pi})$. In our CTC model, this is equivalent to taking the argmax of each output in the time series and removing the gaps.

An alternative heuristic search method is beam search, which has been used extensively in the decoding of neural networks, including CTC models [20]. Beam search iterates through the output y, keeping a fixed-size list (or 'beam') of the best solutions. The size of this list is parameter termed the beam width, and represented with W. The algorithm is the following: for each iteration t in $\{1..T\}$, update its probability at time t using the forward recursions. Next, extend each label in the beam by each character in the alphabet $\{A, C, G, T\}$ and calculate the corresponding forward probabilities. Finally, prune the beam down to the W labels with the highest probabilities. By increasing the beam width W, more solutions are tracked at every iteration.

Pair decoding

Assuming the independence of each read, and a flat prior over labels, the best label is given by

$$\hat{\ell} = \text{argmax}_{\ell} P(\ell|y_1, y_2) = \text{argmax}_{\ell} P(\ell|y_1) P(\ell|y_2)$$

We extend our beam search to work in two dimensions over the $T_1 \times T_2$ space of both reads. Much as in each iteration of a standard beam search, the extensions of each sequence are added to the beam and the score of each sequence is updated. Finally, the beam is pruned down to the top W hits with the highest scores, where W is the beam width. We present below two variations of the algorithm.

In the first, we iterate over $t_1 \in 1..T_1$, and at each step update the forward probabilities for labels in the beam at t_1 (read 1) and for $t \in 1..T_2$ (read 2). Each label in the beam is assigned a score equal to the forward probability from read 1 at t_1 times the maximum forward probability from read 2 in the range $1..T_2$.

$$\text{score}(\ell) = \alpha_1(t_1, |\ell|) + \max_{t \in 1..T_2} \alpha_2(t, |\ell|) \tag{2.1}$$

We also tested a second symmetric variation, where each iteration of the search considers both a column and a row of signal space, and the beam scores are weighted by the respective maximum forward probabilities in those ranges. At each iteration both t_1 and t_2 are incremented by one.

$$\text{score}(\ell) = \max_{t \in t_1..T_1} \alpha_1(t, |\ell|) + \max_{t \in t_2..T_2} \alpha_2(t, |\ell|) \tag{2.2}$$

Both of these strategies are implemented in PoreOver and yield similar accuracies on our test data. The second method reduces the number of beam update steps and runs comparatively faster when adapted to use the alignment envelope, and so was used for the accuracy benchmarks in Figure 2.2 (and is the default setting of PoreOver).

Because the complementary strand passes through the pore with high probability but not every read generates a second strand, there is an additional step during sequencing to determine whether two reads are complementary strands or not. For this work we just relied on the determination of the Guppy basecaller. While one could in theory train two RNNs to operate on the forward and reverse strands and then run consensus on those two probabilities, here we adopt a simpler approach by taking the "reverse complement" of the softmax probabilities, for example:

$$\begin{array}{c|ccc|} t= & 1 & 2 & 3 \\ \hline A & 0.5 & 0 & 0.5 \\ C & 0 & 0.6 & 0.5 \\ G & 0 & 0.2 & 0 \\ T & 0 & 0 & 0 \\ \epsilon & 0.5 & 0.2 & 0 \end{array} \xrightarrow{\text{reverse complement}} \begin{array}{c|ccc|} t= & 1 & 2 & 3 \\ \hline A & 0 & 0 & 0 \\ C & 0 & 0.2 & 0 \\ G & 0.5 & 0.6 & 0 \\ T & 0.5 & 0 & 0.5 \\ \epsilon & 0 & 0.2 & 0.5 \end{array}$$

For Bonito decoding, we modified the basecaller (version 0.2.2) to save softmax probabilities and adapted our decoding algorithm to work with a CTC model that merges repeated characters.

Pair decoding can be trivially parallelized over multiple reads for a large speedup. For our Bonito experiment, decoding the 5,000 read pair test set using 20 parallel processes on an Intel i9-9820X workstation took 17:38 minutes, yielding an average decoding speed of 1,628 consensus bp/s/thread.

Our pair decoding algorithm makes use of a very narrow alignment envelope defined in signal space using a sequence alignment. Given a match state in the sequence alignment, and the default padding of 5 on either side, we would expect the average size to be 9+5+5 =19. Empirically, we see a mean envelope width of 22, not far off this estimate. For two average reads of 10kb and 90,000 signals, this envelope would constrain the search to just 0.024% of the entire matrix, an efficiency which helps justify our use of a pairwise sequence alignment step.

The quality of the consensus sequence tends to depend on the quality of this sequence alignment, and indeed the main failure mode appears to be cases where the two 1D^2 reads are unalignable. As the algorithm assumes sequences will be globally alignable, any sequences with a significant length mismatch are skipped by PoreOver. Of the remaining reads, a small fraction have poor pairwise alignments that result in poor consensus sequences (i.e. consensus accuracy lower than the mean 1D accuracy of the two reads), sometimes due to one read being of significantly worse quality and bringing the overall average down. Altogether, including the sequences that are skipped, these failure cases make up $< 2\%$ of the our test data.

Pair decoding and genome assembly

We sought to investigate whether the accuracy improvement from consensus decoding would carry through downstream assembly and polishing. To do this, we created two assemblies from our set of 5,000 paired reads. In the first, we ignored the pairing information and treated each read independently, basecalling with the Bonito network and decoding with the Viterbi algorithm. For the second assembly, we basecalled each read with Bonito but then ran PoreOver to generate consensus sequences for each pair. The resulting sets of sequences were assembled with miniasm [37] and then polished for four rounds with Racon [62]. The resulting contigs were compared against the reference and the results are shown in Figure 2.5. We find that while the unpolished assembly of $1D^2$ reads is significantly more accurate, this difference is reduced after several rounds of polishing. Thus, this highlights that the main use of pair decoding is for maximizing single-molecule accuracy, and has only a marginal benefit when used in an assembly+polishing workflow.

Neural network architecture and training

Our example basecalling network, PoreOverNet, consists of a single convolutional layer followed by three bidirectional GRU [8] layers (Figure 2.3). Output is passed through a softmax function to yield probabilities for each nucleotide plus a gap character, $\{A, C, G, T, \epsilon\}$. Under this model, the gap character represents no change in the pore, and so the output probability trace consists of peaks of probability as each nucleotide passes through the pore, followed by stretches with high gap probability (Figure 2.1A).

Given that training organism influences the generalizability of the network [60], we sought to include a broad taxonomic diversity. A training set of R9.4 reads was assembled from a sample of 10,000 human reads from the nanopore whole genome sequencing consortium [26] and 10,000 microbial reads from the Zymo mock community [42] spanning 8 bacterial and 2 yeast species.

Reads were re-basecalled with the Guppy basecaller (version 3.2.4) and mapped back to reference genomes. Tombo[2] was used to re-align the raw signal to the reference sequence and correct previous basecalling errors. These corrected reads were split into chunks of 1000 measurements and used as the training set. A fraction of the data was held out as a test set

[2]`https://github.com/nanoporetech/tombo`

and used to evaluate the performance of our model during training. The neural network was trained for ten epochs using the Adam optimizer [31] (learning rate of 0.0005) to minimize CTC loss.

Evaluating decoding algorithms for single read basecalling

In addition to the read pair decoding presented in the main text, we also compare single read decoding algorithms using both our own trained basecalling network (Figure 2.3), as well as ONT's basecaller Guppy, which implements a variant of CTC called "flip-flop". The same test set of 10,000 reads was used as in the previous benchmark, though the pairing information was ignored and reads were treated as standard 1D reads.

Simplified CTC model

Test reads were split into chunks of 1000 measurements, batched together, and fed through PoreOverNet to yield the softmax probabilities. Decoding was then done with both (1) Viterbi best path and (2) a beam search (Figure 2.4A). The use of beam search over Viterbi yielded a very slight improvement in median accuracy from 87.6% to 88.1%. Interestingly, the beam width had little effect on the overall accuracy, with $W = 50$ yielding nearly identical results. While beam search does yield a slight improvement over Viterbi decoding, it comes at a disproportionately greater computational cost.

Flip-flop CTC model

While the earliest nanopore basecallers focused on the task of predicting a sequence of k-mers [53], the production basecaller Guppy along with the research basecaller Flappie introduced a character level, CTC-style model known as "flip-flop". The flip-flop model is an adaptation of CTC for the purpose of better calling homopolymers, a known error mode in nanopore sequencing.

The flip-flop model does not use gaps, as in the standard CTC model, but instead has two sets of "flip" (+) and "flop" (-) states, $\{A^+, C^+, G^+, T^+, A^-, C^-, G^-, T^-\}$, with transitions within flip and flop states only emitting a blank character, ϵ. Furthermore, transitions from flip to flop states are only allowed between the same nucleotide (e.g. $A^- \rightarrow A^+$). Internally, the basecaller generates a transition matrix for each time step and then runs a Viterbi decoding to generate the final basecalled sequence. The marginalized version of these transition probabilities are stored in FAST5 files, allowing us to use our own decoding algorithms on the flip-flop probabilities. However, we need to adapt the calculation of the forward probabilities to take into account the flip and flop states:

$$\alpha^+(t,s) = \alpha^+(t-1,s)\cdot y(t,\ell_s) + \begin{cases}\alpha^-(t-1,s-1)\cdot y(t,\ell_s) & \text{if } \ell_s=\ell_{s-1}\\ (\alpha^+(t-1,s-1)+\alpha^-(t-1,s-1))\cdot y(t,\ell_s) & \text{otherwise}\end{cases}$$

$$\alpha^-(t,s) = \alpha^-(t-1,s)\cdot y(t,\ell_s+4) + \begin{cases}\alpha^+(t-1,s-1)\cdot y(t,\ell_s+4) & \text{if } \ell_s=\ell_{s-1}\\ 0 & \text{otherwise}\end{cases}$$

Results from running on the same set of 10,000 reads are shown in Figure 2.4B. Surprisingly for the flip-flop model, the Viterbi algorithm actually outperforms the beam search on this data. The Viterbi decoding yielded a median accuracy of 90.9% while beam search decoding yielded a median accuracy of 88.9%. Despite the sequences returned by the beam search having higher probabilities, they tend to be less accurate when aligned to the reference genome. Likely because of this, our pair decoding algorithm, which is a form of beam search, does not work well with flip-flop basecalling.

For some reason, it appears that the more probable sequences aren't necessarily more accurate. While it is unclear why this was the case with the flip-flop model but not our simplified CTC model, it could be symptomatic of the way these models are trained and evaluated. Much as in several natural language processing tasks, there is a mismatch between the maximum likelihood objective used in training, the CTC loss, and the actual metric used for evaluation, the edit distance or alignment accuracy between the predicted and true labelings. While this metric is discrete and non-differentiable, there has been some success in speech recognition using techniques from reinforcement learning to approximate this gradient and optimize the edit distance directly [20]. Applying policy gradient style approaches that maximize the reward function (in this case alignment accuracy) to CTC basecalling could be an avenue for future research.

In addition to the "flip-flop" model available in the production basecaller Guppy, ONT have also recently introduced another basecalling paradigm known as "run-length encoding", which is implemented in the research basecaller Runnie. Under the run-length encoding model, the neural network outputs the best nucleotide as well as parameters of a discrete Weibull distribution which characterizes the length of the repeat. While this makes single read decoding trivial (by predicting the mode of the parameterized distribution), the 2D beam search described could be adapted to work for run length encoded output. Indeed, one of the strengths of beam search is the ease with which it can be adapted (e.g. to use a language model in speech recognition).

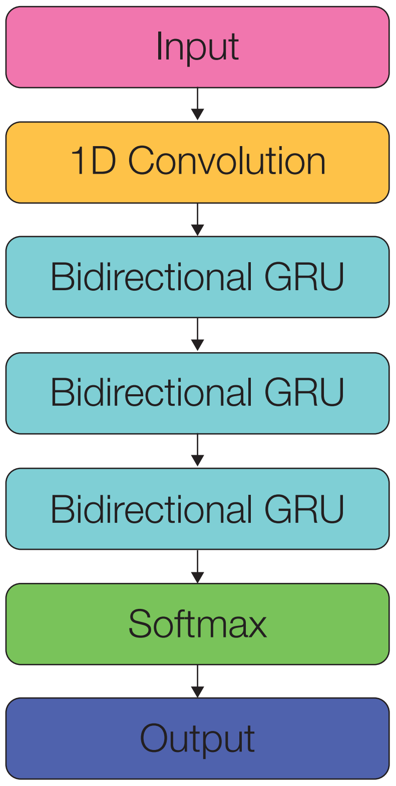
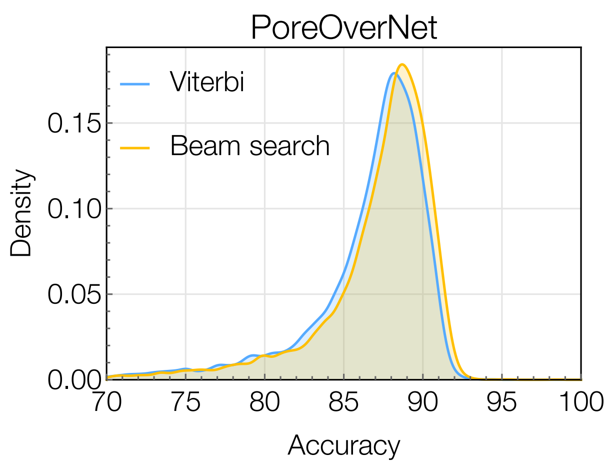


Figure 2.3: Architecture of PoreOverNet.

Input is processed with a single convolutional layer (256 filters, kernel of length 9, stride of 1) followed by three stacked bidirectional GRU layers (128 units each). The model has 893,189 parameters in total. The softmax output is fed into a CTC loss function, and minimized during training.



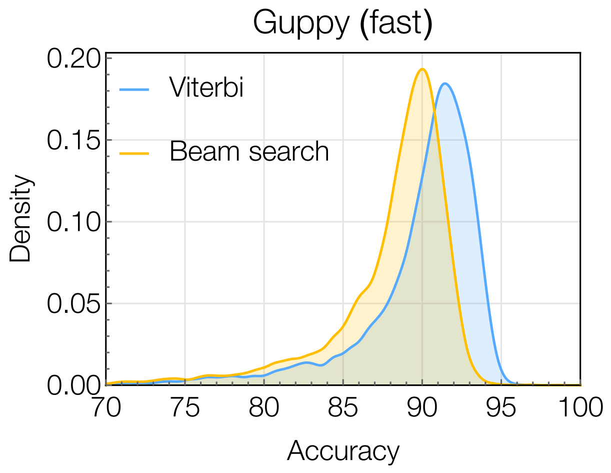


Figure 2.4: Single read decoding accuracy.

(A) Comparison of single read decoding algorithms using output from our network PoreOverNet, which implements a simplified CTC model that does not merge repeated characters. (B) Single read decoding algorithms run on output of the Guppy basecaller, which implements a variation of CTC for calling homopolymers called “flip-flop”. In this case the beam search surprisingly returned lower accuracy basecalls than Viterbi decoding. A beam width of 10 was used for both plots.

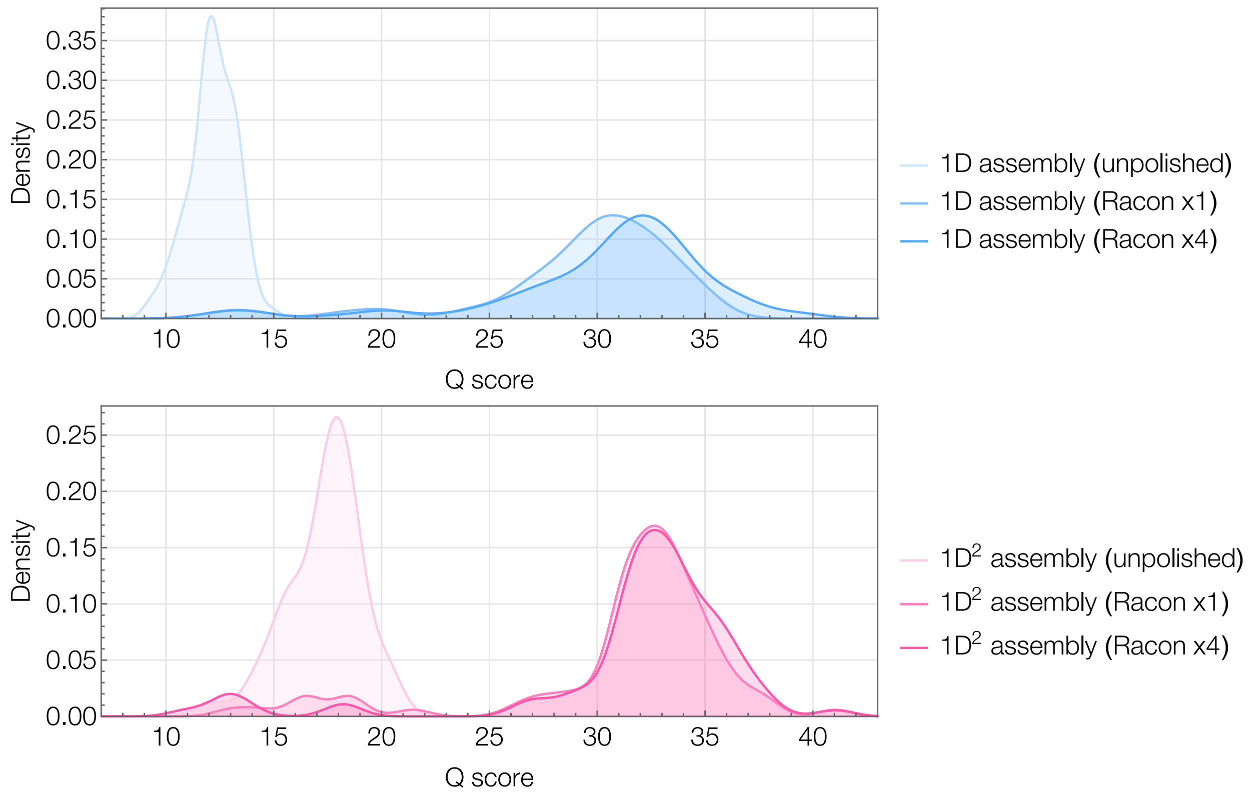


Figure 2.5: Accuracy of assemblies generated with and without consensus decoding of read pairs.

After initial assembly with miniasm, assemblies were polished for several rounds with Racon. The Q-score is defined as $-10\log(\text{error rate})$.

Chapter 3

Nanopore consensus decoding improves accuracy of amplicon sequencing at low read depth

3.1 Introduction

Nanopore sequencing, such as that done on the Oxford Nanopore Technologies platform, allows for long-read single-molecule DNA sequencing. Since the first generation of the MinION sequencer, the single read error rate has fallen from over 20% [53] to 1-5% [55]. Despite these great strides in basecalling accuracy, this error rate remains high relative to other next-generation sequencing techniques. Higher accuracies are only possible through consensus approaches, where higher depth sequencing is used to average out errors across individually noisy reads.

The most common scenario arises in whole genome sequencing, where overlapping reads are aligned and a rough draft assembly sequence found. More targeted methods for high accuracy amplicon sequencing include R2C2 [64], which uses rolling circular amplification to generate multiple copies of the target sequencing. Alternatively, by tagging different amplicons with unique molecular identifiers (UMIs), reads can be clustered by barcodes and then fed into a consensus algorithm such as Racon [62].

Once an initial consensus has been generated from these clustered reads, the accuracy can be further increased through polishing. Polishing involves re-aligning the reads to this draft consensus sequence and applying an additional algorithm to correct lingering sequencing errors. Mirroring the shift in basecalling methods, polishing was originally done with hidden Markov models such as nanopolish [38], and now almost entirely by neural networks such as Medaka [46] and HELEN [57].

While most existing polishing algorithms work in sequence space, using reads already basecalled, we sought to develop a polishing method to use the intermediate output of the neural network basecaller.

For the large class of basecallers that are trained using connectionist temporal classification [21], the output of the neural network defines a probability distribution over sequences. This necessitates the further step of *decoding*, which involves finding the single sequence which maximizes this probability. Finding this optimum exactly is not computationally tractable, so inexact algorithms such as beam search are used. Viterbi decoding finds the highest probability path, which is often close to, but not necessarily the optimal sequence.

Following the success of our previous approach using two reads (Chapter 2, [59]), we seek to use these basecaller probabilities for the more general case of decoding multiple reads. The two read consensus problem is a special case motivated by the ability to read both complementary strands of a DNA molecule using special protocols (the current Duplex method, similar to previous 2D and 1D^2 protocols). For two reads, a substantial gain in accuracy is possible by storing the output of the neural network and running a pair decoding algorithm, a beam search modified to work over the space of two aligned reads. Consensus decoding of these probabilities for the case of two reads reduced the error rate by more than half [59], and has since been adopted into ONT's software for duplex calling (`https://github.com/nanoporetech/fast-ctc-decode`).

In this work, we extend our consensus algorithm to the more general case of N reads, and test it on UMI-tagged amplicon sequencing data [30]. We compare with Medaka [46], a neural network polisher developed by Oxford Nanopore Technologies which operates only in sequence space. Using R9.4.1 data, we find four reads are sufficient for Q25 accuracy with our polishing approach, which roughly halves the error rate at that depth from using Racon alone or in conjunction with Medaka.

As consensus basecalling comes with an added computational cost, it is best designed for situations of low read depth, either rolling circular amplification or selective use in UMI amplicon workflows to bins with insufficient reads for other deep learning polishing methods.

3.2 Methods

Preliminaries

Connectionist temporal classification was originally developed for speech recognition and has been successfully applied to nanopore sequencing by a variety of basecallers.

The CTC output probabilities y are a $T \times 5$ matrix, where the tth row represents the emission probabilities at time t of the four nucleotides plus a gap character (ϵ).

These CTC probabilities can be interpreted as the emission probabilities of a linear hidden Markov model [58], where the probability of a path, π, through this profile is the product of the individual probabilities

$$P(\pi|y) = \prod_{t=1}^{T} y(\pi_t, t)$$

Sequences longer than T have zero probability.

The probability of a label sequence ℓ is obtained by the sum over all paths that would generate that label (after removing the gap characters) and can be efficiently computed with dynamic programming [16]. Analogously to HMMs, we define the Forward probability of a label as $\alpha(t, s)$, the probability that the first s characters of ℓ having been emitted by position t of the underlying HMM.

The core recursion is,

$$\alpha(t, s) = y(t, \ell_s)\alpha(t-1, s-1) + y(t, \epsilon)\alpha(t-1, s)$$

terminated by the base case $\alpha(0, 0) = 1$. The probability of the full sequence is thus $P(\ell|y) = \alpha(T, |\ell|)$.

The Viterbi decoding of each read establishes a read sequence-to-signal mapping (technically sequence-to-output), while the alignment of each read to the draft consensus sequence establishes a read sequence-to-draft sequence mapping. Combining these two, we can define $\Phi_n(i)$, which returns the range of signal in read n that is aligned to base i of the draft consensus sequence.

Beam search polishing

In the case of pair decoding [59], both reads are basecalled individually using a fast decoding algorithm like Viterbi, and the pairwise alignment of the resulting sequences is used to constrain a beam search over the basecaller probabilities in 2D space. Generalizing this approach to N reads would require construction of a similar alignment envelope in N dimensional space. In this work, we adopt a more tractable approach by using the draft assembly sequence as an anchor for the beam search iteration. Let the beam be B, a set of sequences that are tracked during search. The beam search iterates over positions in the draft assembly, d, calculates a score from the aligned regions of all reads, and prunes down to the top W sequences, where W is the beam width. Note that only the alignment of reads to draft is used by the beam search and that the identity of the draft base does not enter the score calculation. The beam polishing algorithm is described by the following pseudocode:

```
for $i \in 1..|d|$ do
    for $\ell \in B$ do
        update Score($\ell$);
    end
    for $c_1 \in \{A, C, G, T\}$ do
        for $c_2 \in \{A, C, G, T\}$ do
            update Score($\ell + c_1 + c_2$);
        end
    end
    Trim $B$ to top $W$ sequences.
end
```

The score is calculated by looking up the signal regions where each read is aligned to a given base of the draft assembly,

$$\text{Score}(\ell) = \sum_{n=1}^{N} \max_{t \in \Phi_n(i)} \log \alpha_n(t, |\ell|)$$

where α_n is the Forward probability using the output of the nth read.

Two extension steps are used per iteration; this allows the beam search to consider sequences that are longer than the reference sequence. When comparing sequences of various lengths, longer sequences will tend to have lower scores because of multiplying more small probabilities. Inspired by approaches in machine-translation [67], we optionally use a simple length normalization scheme to address this:

$$\text{Score}_{\text{LN}}(\ell) = \frac{\text{Score}(\ell)}{|\ell|}$$

This beam polishing algorithm is included our basecalling and consensus software PoreOver (`https://github.com/jordisr/poreover`), which is freely available under an MIT license.

Benchmark dataset

The ribosomal RNA (rRNA) dataset of UMI-tagged reads was used from Karst et al [30]. Reads were clustered by UMI in an error-tolerant manner, using the authors' protocol. A random subset of 1000 read bins was chosen and used as the basis for benchmark studies. Each of these UMI bins was randomly downsampled to include 2 to 15 reads, which was then used to generate consensus sequences.

An initial median sequence was obtained with USEARCH [17], and three rounds of Racon were done to produce a draft consensus sequence. Reads were then aligned back to this draft consensus sequence with minimap2 [37]. This alignment of reads to the draft, as well as the CTC basecaller probabilities were used as inputs to the beam search polishing algorithm.

Basecalling

Basecalling was done with the R9.4.1 model of Bonito v0.2.2, one of ONT's research basecallers. Bonito was modified to save the CTC probabilities output by the forward pass of the network. These probabilities were then loaded into PoreOver for multi-read decoding. As a comparison, polishing was also done using ONT's Medaka (v0.11.5) with the `r941_min_high_g344` model.

Evaluation of accuracy

Final consensus reads were aligned to the microbial references developed in [30] using minimap2 [37]. Accuracy is defined as the number of matches divided by the length of the alignment,

$$\text{Identity} = \frac{N_{\text{matches}}}{N_{\text{matches}} + N_{\text{mismatches}} + N_{\text{insertions}} + N_{\text{deletions}}}.$$

Alignment accuracy was calculated by parsing the CS string of the read-to-reference alignment. In this work we often represent alignment accuracy as a logarithmic Q-score, where

$$\text{Q-score} = -10 \log_{10}(1 - \text{Identity})$$

For example, a Q-score of 20 corresponds to an accuracy of 99%.

3.3 Results

Consensus accuracy increases with read depth

A goal of this study is to quantify how different consensus approaches perform as a function of read depth. To do this, we downsampled UMI bins (see Methods) and ran consensus methods at depths of 2 to 15 reads. Three rounds of Racon were done initially, before polishing with either the neural network Medaka or our beam polishing method (implemented in PoreOver).

As expected, consensus accuracy rises with read depth (Figure 3.1 and 3.2). Beam polishing outperforms Racon alone at all read depths, and outperforms Medaka at depths of less than 10 reads. The marginal benefit of polishing using basecaller probabilities is most pronounced at lower read depths. Indeed, at read depths of 5 and below, beam polishing is roughly halving the median error rate of Racon (e.g. from 0.6% to 0.3% at a depth of 4). Below 5 reads, Racon actually surpasses in Medaka in accuracy, while at greater read depths, Medaka performs better. At 10 reads, Medaka performs similarly to PoreOver, though at a much smaller computational cost. Beyond 10 reads, the median Q-scores are similar between PoreOver and Medaka, though Medaka has a higher fraction of reads without any errors (Figure 3.1B).

These three consensus methods all work differently, and have correspondingly different error profiles (Figure 3.3). Deletions represent the greatest source of error for PoreOver as well as Racon, and persist even at higher read depths. Medaka, on the other hand, makes relatively fewer deletion errors, with insertions being more common, especially at higher read depths. In the case of PoreOver, roughly 40% of the deletion errors come from homopolymers regions, a known source of error with nanopore sequencing. As such, the lengths of homopolymers are systematically underestimated, particularly for homopolymers over 5 bases (Figure 3.4).

Sensitivity to beam search parameters

As our beam polishing method just uses the probabilities output by the upstream basecalling neural network, it does not require any training nor learning parameters from data. Nevertheless, there are hyperparameters used in our approach, namely the beam width and the use of length normalization. We test the sensitivity of the final accuracy to these hyperparameters (Figure 3.5) and find that larger beams and length normalization both increase the accuracy of beam polishing. For the beam width, this comes with a linear increase in the run-time of the algorithm. We use a beam width of 25 as the default for the rest of this work.

3.4 Discussion

Given the relatively high error rate of nanopore sequencing compared to other next-generation sequencing techniques, applications that require high levels of accuracy require consensus approaches. While most existing methods for polishing work in sequence space, prior work with pair decoding showed the benefits that come from working with the raw basecaller probabilities directly.

In this work we generalize the pair decoding algorithm to decode a high accuracy consensus sequence from any number of reads aligned to a draft consensus sequence. We present a beam search algorithm do this multi-read decoding task. Working with intermediate basecaller probabilities comes at an added computational cost, and this beam search approach scales linearly with the read depth. We find that at read depths over 10, the benefit of doing beam polishing compared to a neural network polisher such as Medaka is negligible, despite the added computational cost. At lower read depths, however, beam polishing is able to maximize the accuracy of the basecalling and find consensus sequences consistently better than Racon or a combination of Racon and Medaka. The general conclusion of this work is that at low read depth, additional accuracy can be had by working in basecaller probability space. While we observed the crossover point around 10 reads for beam polishing vs neural polishing, this could be dependent on additional factors, such as pore chemistry (we used R9.4.1) and basecaller version (Bonito v0.2.2), and the resulting single read accuracy.

Finally, while the neural polisher Medaka performed poorly at low-read depth in this study this likely reflects a much higher average read depth in the training data for Medaka. It is thus possible that a neural network could be trained to polish assemblies more effectively at low read depth. That being said, while our algorithm depends on a few hyperparameters (i.e. beam width) which have been tuned in this study, it does not explicitly learn any parameters from data and so does not have similar issues with training dataset size and/or diversity.

As Oxford Nanopore Technologies moves towards re-reading the same molecule[1], these types of low read depth consensus methods will become even more relevant.

3.5 Acknowledgments

This work was supported by NIH/NCI grant CA220441, NIH/NHGRI training grant T32 HG000047, and by Oxford Nanopore Technologies.

[1]Clive Brown, London Calling 2021 , https://www.youtube.com/watch?v=AlAnmvbNwfY

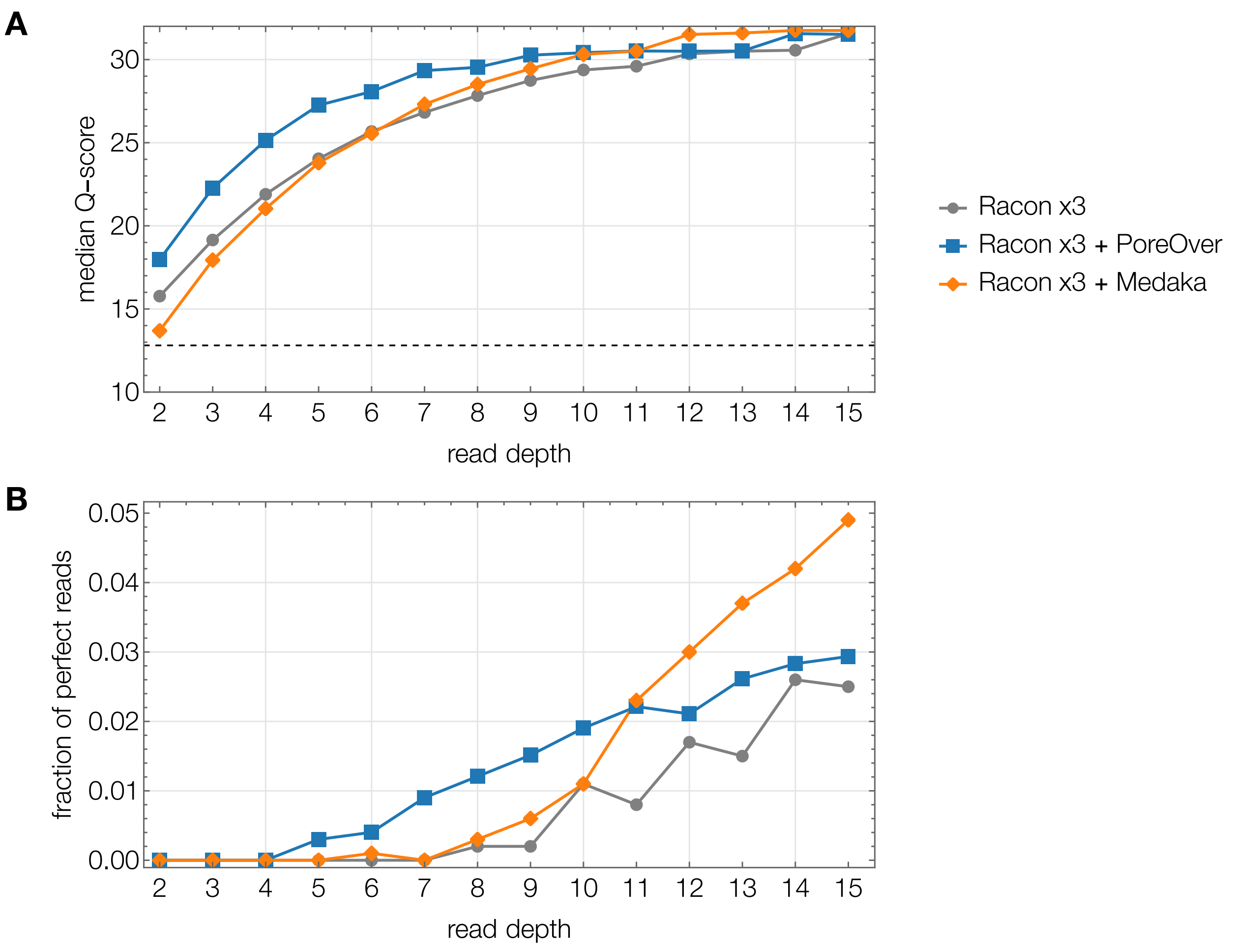


Figure 3.1: Consensus accuracy increases as a function of read depth.

(A) Beam polishing consistently outperforms Racon alone, though the marginal benefit decreases with increasing read depth. Interestingly, below 5 reads Medaka under-performs Racon alone. At read depths of 10 and higher, the accuracy Medaka is roughly equivalent to the beam polishing, while being much less computationally demanding. (B) Fraction of perfectly called reads without errors is also an important metric. PoreOver surpasses Medaka up to 10 reads, at which point neural network polishing generates more perfect reads.

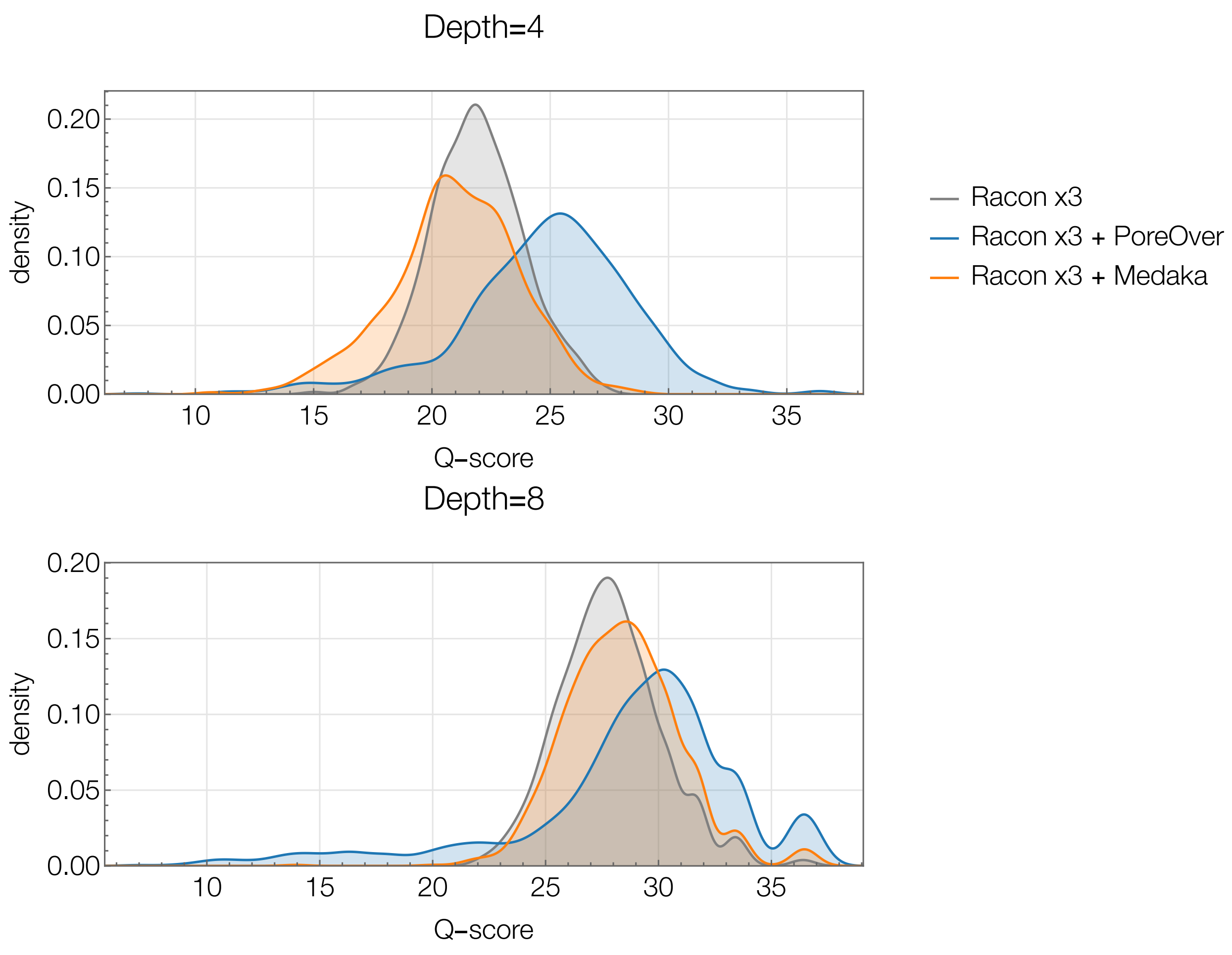


Figure 3.2: Accuracy distributions at two different read depths.

Unsurprisingly, higher read depths allow for higher accuracy consensus sequences. The relative benefit of beam search polishing is most pronounced at lower read depths, as shown here with read depth 4 and 8.

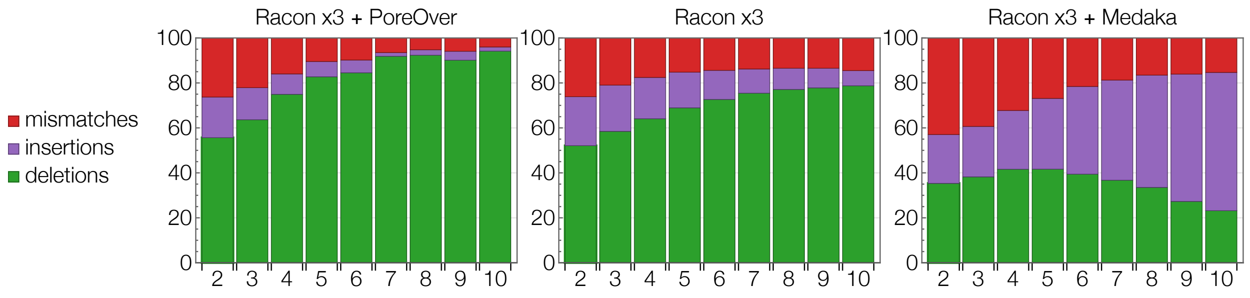


Figure 3.3: Error profiles differ across tools and across read depths.

For beam polishing and Racon, deletions are the predominant source of error. At higher read depths, a much smaller number of insertions and mismatches are made. Medaka, however, makes proportionally more insertion errors, which actually increases with read depth.

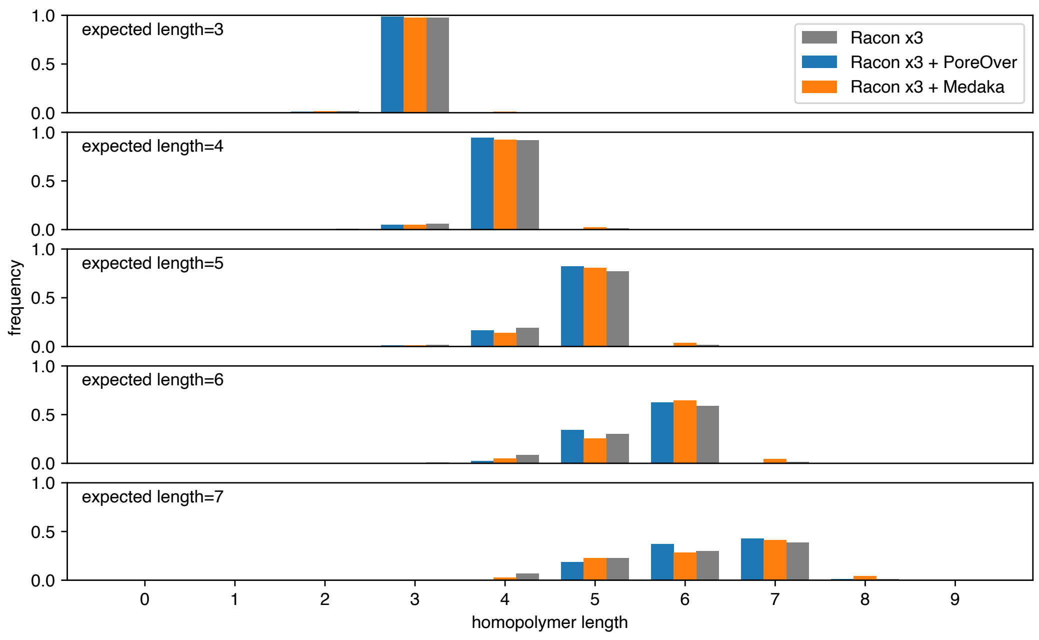


Figure 3.4: Homopolymer length distribution after polishing.

Difference between expected and basecalled homopolymer length shown for reads downsampled to a depth of 4. Deletions within homopolymers are a predominant source of error at this depth for all polishers tested. While all polishers yield a mode of expected homopolymer length, the accuracy falls as the length of homopolymer increases.

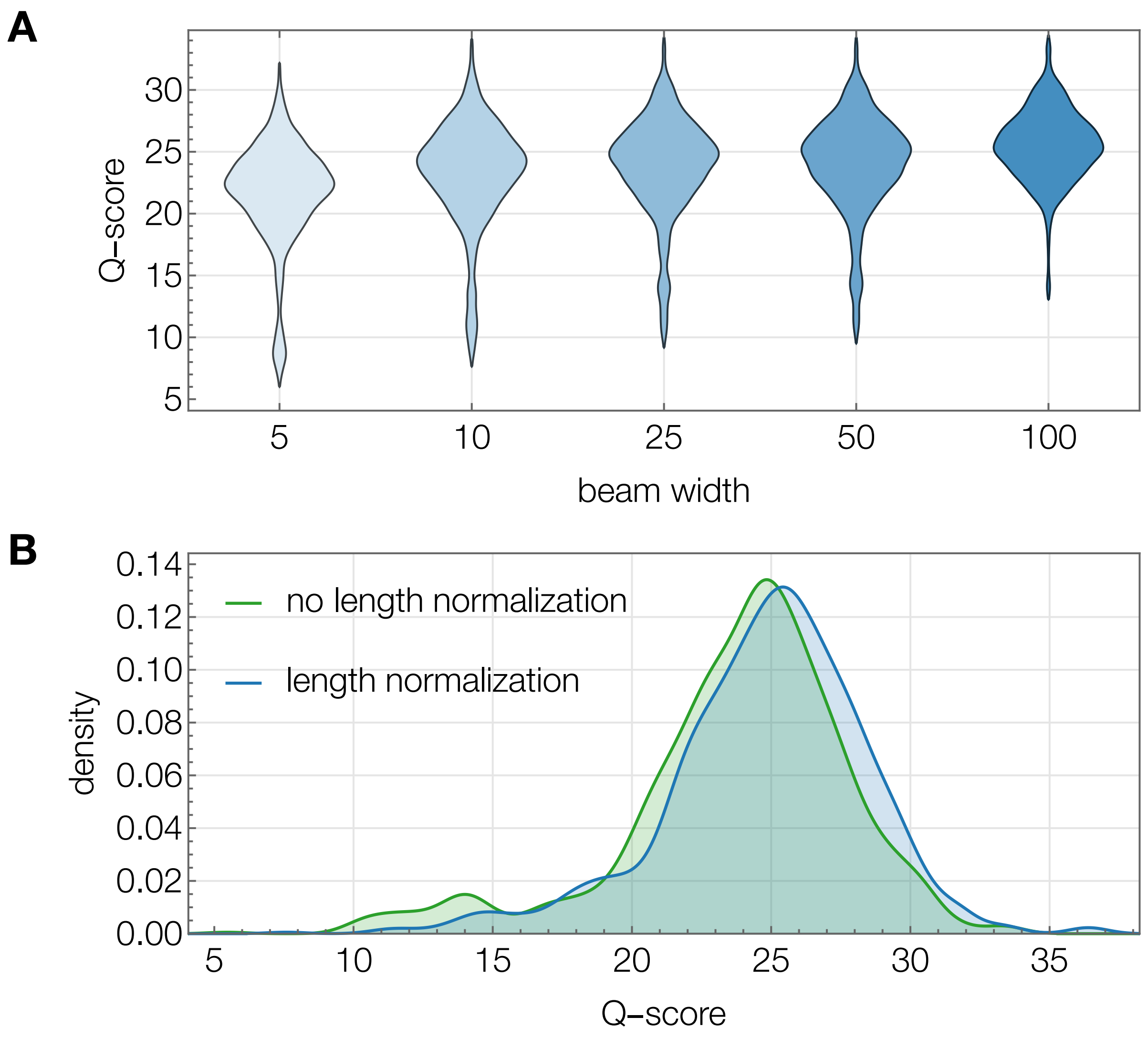


Figure 3.5: Sensitivity of the beam search to hyperparameters.

(A) The main hyperparameter is the beam width, which is the number of candidate sequences that are tracked and extended during the search. Accuracy increases with increasing beam width, but at increased computational cost. (B) We explore a simple length-normalization scheme, where scores of sequences on the beam are scaled by sequence length. This yields a small improvement in accuracy (blue distribution).

Chapter 4

Neural network polishing from raw nanopore signal

4.1 Introduction

The sequencing platform developed by Oxford Nanopore Technologies (ONT) allows for long-read single-molecule sequencing on a portable device. While advancements in basecalling algorithms and sequencing chemistry have brought the error rate down significantly from the earliest sequencing kits [53], the error rate still remains high relative to next-generation short-read sequencing. Higher accuracy sequences can be obtained by combining multiple overlapping reads into a single consensus sequence. These overlapping reads could come from sequenced amplicons [30][64], or whole genome sequencing and assembly [42]. After finding the overlap between reads and generating an initial rough draft assembly or consensus sequence, additional gains in accuracy can often be had by re-examining the reads as they align to the draft sequence. This practice is referred to as polishing, and involves re-aligning the reads back to the draft assembly, and running a separate algorithm in an attempt to refine the draft sequence. Much as neural networks replaced hidden Markov models in nanopore basecalling, as similar shift has happened with polishing. Nanopolish [38] is an HMM polisher which saw much initial success, but has since been superseded by polishers relying on neural networks.

One such neural network polisher is Medaka [46], which is developed by Oxford Nanopore Technologies. It uses the counts of bases in each column of the read pileup as input features to a recurrent neural network. While Medaka uses only these summary statistics, other polishing networks have been developed which use the full read pileup as input. For example, NeuralPolish [25] uses recurrent neural networks over both the columns and rows of the read pileup. Additionally, instead of single nucleotides, it predicts homopolymers of 1-5 bases. This allows the network to correct homopolyer deletions and gives it the ability to output sequences longer than the draft sequence. Other approaches such as HELEN [57] take a genome assembly graph as input for polishing.

For PacBio sequencing, which can generate higher accuracy consensus sequences from circular sequencing (HiFi), the recent DeepConsensus [1] achieves state-of-the-art performance using a transformer architecture with the read pileup as well as features from the raw PacBio signal. In Nanopore sequencing, the ONT tool Remora [47] also uses raw signal in a post-basecalling pipeline to call variants and DNA modifications. Inspired by these approaches that use raw signal features in addition to the pileup of basecalled reads, in this work we develop and test a neural network polisher for nanopore sequencing. Our polishing network takes as input the raw nanopore signal in addition to the draft sequence and the pileup of basecalled reads aligned to this draft. We test the effectiveness of this approach on microbial genome assemblies and conduct ablation experiments to explore the effect of including raw signal on neural network polishing for nanopore sequencing.

4.2 Methods

Data input and preprocessing

Polishing requires basecalled reads in FAST5 format as well as a draft sequence, such as one generated from an assembly tool like Flye [32] or Miniasm [37]. These reads must be re-aligned to the draft assembly sequence to generate a read pileup.

The basecalled FAST5 reads include the raw measurements of electrical signal generated as the DNA strand moves through the pore, as well as a move table, which defines a segmentation of the raw signal into windows corresponding to each base. As sequencing runs can generate numerous FAST5 files, it is not feasible to load all into memory simultaneously for preprocessing. To this end, we implemented a least recently used (LRU) caching system to read FAST5 files into memory as needed during featurization.

The read-to-draft alignments from this read pileup are converted into a multiple alignment by adding sufficient gap characters to cover indels with respect to the draft sequence. Each element of this multiple sequence alignment is mapped to an 11-letter alphabet consisting of the four nucleotides mapping to the forward strand, four nucleotides mapping to the reverse strand, and two special characters differentiating between insertions and deletions with respect to the draft sequence. An additional character representing no alignment is also used. The segmented raw signal and preprocessed multiple alignment of reads-to-draft are split into 64bp windows along the draft assembly and then fed into the neural network.

Due to GPU memory limitations for the neural network, some additional size constraints are imposed on the input features. Each raw signal window is truncated to a maximum of 80 measurements, which affects only a very small fraction of bases (<0.2%). Additionally, only the first 50 reads are included from each pileup, which artificially caps the read depth of the input.

Network architecture

Each variable-length segment is passed through an identical signal embedding block, which consists of a 1D convolutional layer followed by a single bidirectional GRU (Figure 4.1). The final state of the GRU, which has the size d_{signal}, serves as a fixed-length summary of a single signal segment.

In parallel, the read pileup is passed through a single embedding layer that maps each letter in the 11-letter expanded input alphabet into a fixed-length vector of size d_{seq}. This same embedding layer is also applied to the draft assembly sequence. For each element of the original pileup, the corresponding fixed-length embeddings are concatenated to yield an embedding of size $(d_{\text{signal}} + d_{\text{seq}})$.

The original $m \times n$ alignment is then passed through n_{cols} GRU layers operating over columns of this alignment. The final GRU state of size d_{encoding} is taken as a summary of each column of the alignment. Finally, another set of n_{rows} GRU layers operates over these summaries to generate a row encoding of the same dimension. The draft assembly sequence

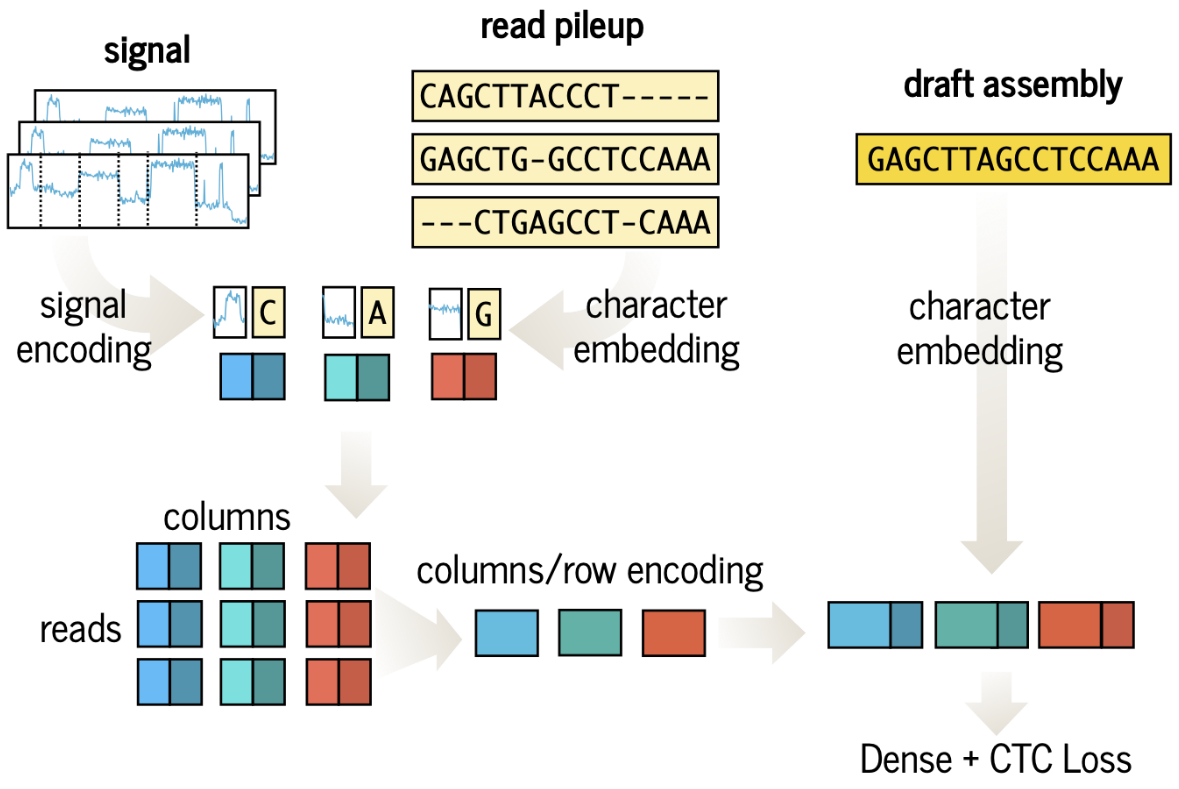


Figure 4.1: Overview of network architecture for polishing with raw nanopore signal.

embeddings are concatenated to this encoding to generate the final internal representation of the input. This is passed through two fully-connected layers to a final softmax output over 5 classes (4 nucleotides plus a gap character), which is passed to the CTC loss function. For the models trained here, we used the following hyperparameters: $n_{\text{cols}} = 1$, $n_{\text{rows}} = 3$, $d_{\text{signal}} = 128$, $d_{\text{seq}} = 32$, and $d_{\text{encoding}} = 512$.

Generating labeled training data

A publicly available dataset [42] of microbial reads was used to construct the training set. This dataset is comprised of 8 bacterial species from the Zymo mock community, and includes high-quality reference genomes. Reads were first basecalled with Guppy (version 5.0.16), and then binned by organism and downsampled to generate assemblies of varying read depths.

Dataset	ID	Species
Training data	bs	*Bacillus subtilis*
	ec	*Escherichia coli*
	ef	*Enterococcus faecalis*
	lf	*Lactobacillus fermentum*
	lm	*Listeria monocytogenes*
	se	*Salmonella enterica*
Test data	pa	*Pseudomonas aeruginosa*
	sa	*Staphylococcus aureus*

Table 4.1: Microbial species from Zymo mock community used for training and evaluating polisher.

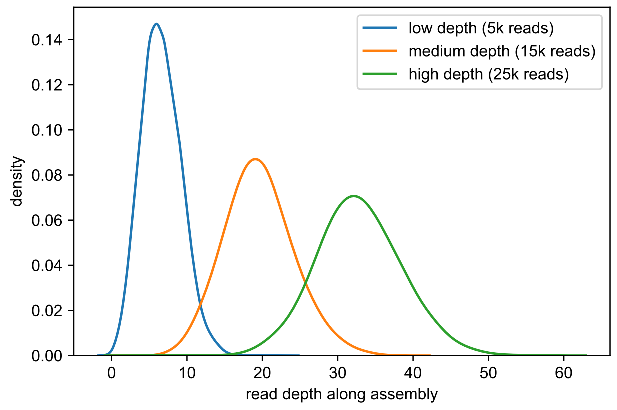


Figure 4.2: Read depth along *E. coli* assemblies used for training.

For training and testing, reads were downsampled to 5000, 15000, and 25000 reads to yield assemblies at varying depths. This is a representative example for the contigs in the *E. coli* assemblies.

Flye was used to generate three assemblies for each taxon starting from 5000, 15000, and 25000 reads, which yielded contigs with varying read depths (Figure 4.2). Draft sequences were aligned to their respective reference genomes, which was used to generate a gold standard training set. These assemblies were featurized and labeled with their corresponding reference bases. After featurization and labeling, three datasets were generated for each of the bacterial taxa. Two taxa (*Pseudomonas aeruginosa* and *Staphylococcus aureus*) were held out as test sets, and the assemblies of the remaining 6 taxa were pooled into one dataset (Table 4.1). Twenty percent of these examples were held out as a validation set to assess training, which left a final training set of nearly 1 million labeled examples.

Training and early stopping

Training was done using the Adam optimizer (learning rate=0.0001) [31] to minimize the CTC loss [21]. At each epoch, loss and edit distance were evaluated on the held-out validation set. Training was stopped after the validation loss failed to decrease for 10 consecutive epochs. A batch size of 16 was used for all models. Training for the model using signal took roughly 120 min/epoch, while the sequence-only models took roughly 40 min/epoch on a single Nvidia 2080 Ti GPU. Weights & Biases [2] was used for experiment tracking.

Availability

Our neural network polisher is implemented in Python 3 and TensorFlow 2 and freely available on GitHub (`https://github.com/jordisr/semapore`).

4.3 Results

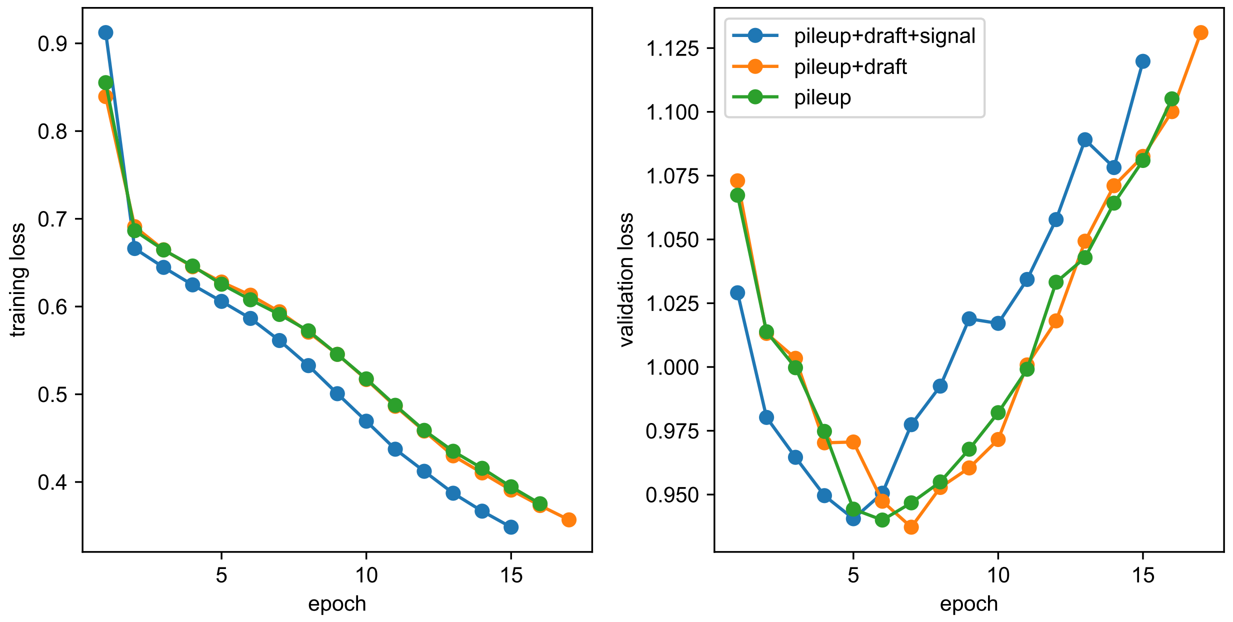


Figure 4.3: Minimization of CTC loss during neural polisher training.

Left panel shows the training loss on training set, which is directly minimized by the optimizer and decreases throughout training. The right panel shows the loss on a held out validation set, which decreases initially and then starts to increase as the network is overfitting to the training data.

Model training

The core network architecture (Figure 4.1) uses the aligned read pileup, the draft sequence, and the raw signal events and is referred to as `pileup+draft+signal`. We additionally trained two ablated versions of this architecture without the raw signal input. One still includes the draft sequence (`pileup+draft`) while the other uses the read pileup alone (`pileup`). Training and validation loss for these architectures are shown in Figure 4.3. While training loss decreases over all epochs, validation loss starts increasing after several epochs, a sign that the network is starting to overfit.

Performance on held-out species

To assess network performance, the model checkpoint from each epoch was evaluated against the *P. aeruginosa* and *S. aureus* datasets, the two taxa held out as a test set (Figure 4.4). For comparison with training data, *E. coli* is also included in the figure. While results are shown for each epoch, this test set was not used for early stopping of training.

After 5-7 epochs, the test error rate on both held-out species stops decreasing, in contrast with the *E. coli* training data, which (expectedly) continues to improve throughout training. This pattern is consistent at all read depths, though interestingly on the 5k test sets performance clearly decreases, whereas on the 15k and 25k datasets this is less pronounced and the error rate simply appears to plateau.

Unfortunately, across all datasets the error rate stays extremely close to the draft error rate, suggesting that model has not learned much beyond outputting the draft sequence. This seems to hold true regardless of architecture, and is even the case for the `pileup` model which was not explicitly provided the draft sequence. The signal-based model appears to outperform the two ablated networks on the sa-5k dataset, which is encouraging, however the improvement relative to the draft is still not enough to make it useful as a polisher.

In this case, the high accuracy of the draft assemblies creates a challenging dataset to train on, as many batches have few to no draft errors to learn on. As an attempt to circumvent this, the training dataset was split into examples with at least a single draft error and those without any, allowing draft errors to be oversampled at training time. Experiments were run with the `pileup+draft+signal` model sampling examples with errors at 25% and 50% rates, though neither of these yielded any improvement on the draft sequence (data not shown).

4.4 Discussion

While the network made negligible reductions in the error rate of the draft assembly, this section covers some of the challenges with developing a polisher for raw nanopore signal.

As the network uses the raw signal, our preprocessing pipeline was designed to work with FAST5 files, the HDF5 format that stores nanopore basecalls along with segmented raw signals. As individual sequencing runs can generate hundreds of gigabytes of raw data, this presents data engineering challenges not seen when working solely with FASTQ basecalls.

For this work, we implemented an LRU caching strategy to store a limited number of FAST5 files in memory, which allowed for a more efficient featurization during data preprocessing.

While a relatively simple RNN architecture was adopted, future work could explore alternative architectures and featurizations: for instance, replacing some of the GRU layers with self-attention layers [63]. Additionally, other featurizations could be explored, such as using the frequency of bases in each pileup column (as Medaka does) rather than the entire pileup. Furthermore, the FAST5 files also include per-base quality scores which could easily be incorporated into the pileup embedding.

Though this work used a single assembler to generate the training data, different tools may yield assemblies with distinct patterns of errors. As such, developing a more universally applicable polishing network might require including additional data from alternative basecallers and assemblers to train a model capable of generalizing. Alternatively, this data could be used to fine-tune multiple models to be assembler and/or basecaller specific.

Lastly, as basecallers (and assemblers) improve in accuracy, errors become rarer and it becomes more challenging to train networks on these rare events. This was evident in our training set, where draft fragments that perfectly matched the reference outnumbered those with at least one error in the draft. This is effectively leads to a class imbalance where many training batches don't have a single error. One reason for the observed poor performance could be getting trapped in a local minimum that just returns the draft sequence. While one potential way of addressing this is to separate training examples into those with and without errors and then oversample error-containing examples at training time, this did not yield an increase in performance, possibly due to distributional mismatch. Taken to an extreme, the diminishing returns from polishing may eliminate the need for a separate step after the initial assembly. With the latest sequencing improvements such as the R10.4 pore, this may already be the case for some bacterial assemblies [55].

4.5 Acknowledgments

This work was supported by NIH/NCI grant CA220441, Oxford Nanopore Technologies, and the NVIDIA Academic Hardware Grant Program.

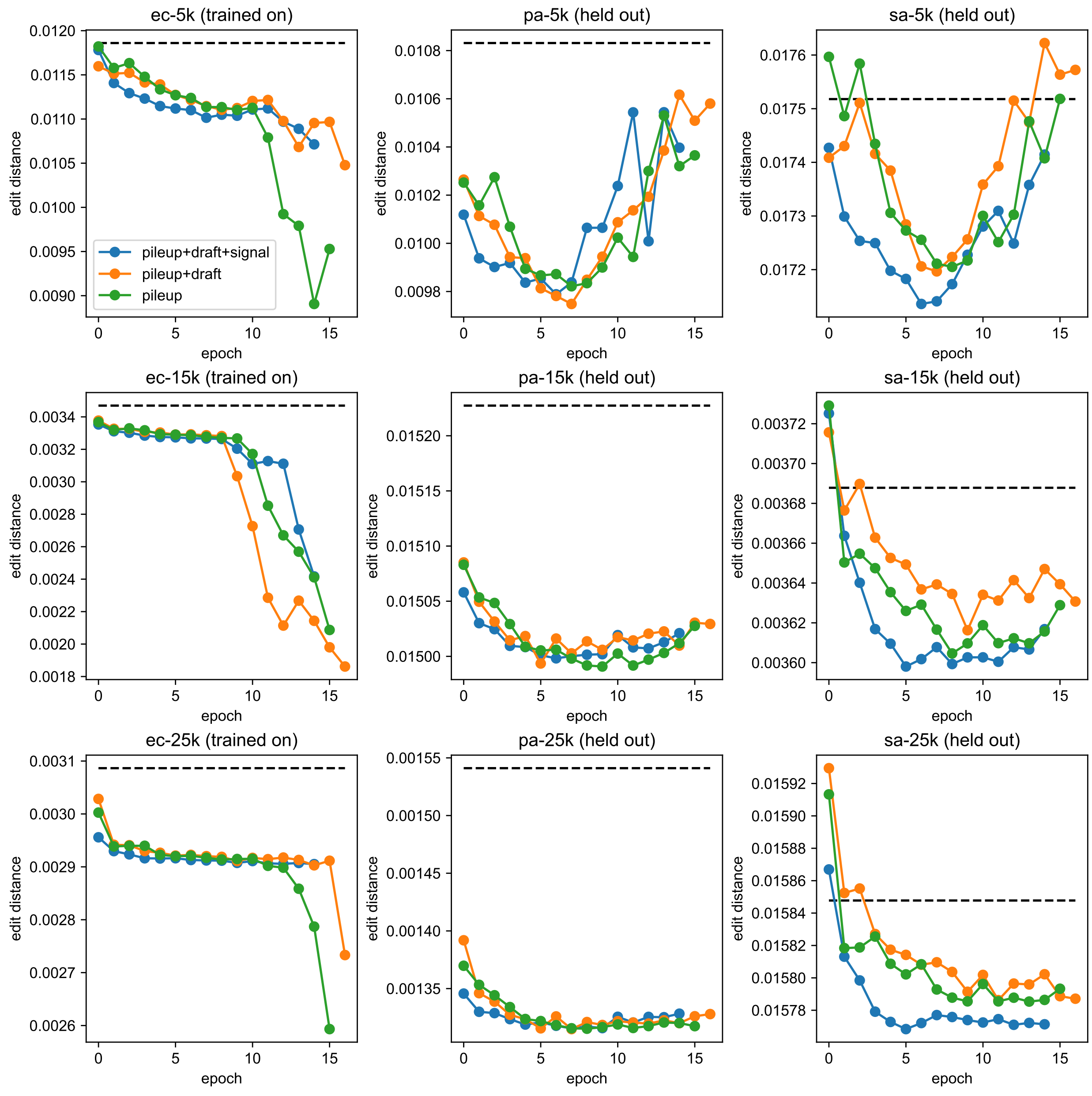


Figure 4.4: Performance of polisher architectures on held-out test species.

Edit distance represents the error rate of the polished assembly and refers to the Levenshtein distance between draft and reference sequences, which is normalized by the length of the reference. The black dashed line shows the mean edit distance of the draft sequence by itself, which is an input to the network. The *E. coli* data in the leftmost column was included in the training set and is included for comparison. Accuracies were evaluated using the model checkpoints saved at each epoch. With the 5k dataset we see overfitting at later epochs. In contrast, with the 15k and 25k datasets, the test error rate appears to plateau very near the draft error rate.

Chapter 5

Policy minimization of basecalling errors in nanopore sequencing

5.1 Abstract

Nanopore DNA sequencing is a promising new technology that uses neural networks to map an electrical current generated by a DNA molecule passing through a pore back to the underlying sequence of DNA bases (A,C,G,T). This step, known as basecalling, shares similarities with the task of speech recognition, and has benefited from advances in that field, such as the use of connectionist temporal classification (CTC) to train the network. Under CTC loss, the softmax output of the RNN describes a probability distribution over all labelings of a certain length. Much as in speech recognition, there is a mismatch between this maximum likelihood loss function used in training, and the edit distance metric used to evaluate accuracy. In the case of speech recognition, policy gradient algorithms such as REINFORCE have been applied with some success to optimize the non-differentiable accuracy metric directly. In this scenario, the probabilistic output of the RNN, which depends on the weights of the network, is intepreted as a *policy*, while the *reward* is the similarity to the true label, in this case one minus the edit distance. In this work, we investigate whether self-critical sequence training (SCST), a variant of REINFORCE, can be used to directly optimize the edit distance and train a basecaller for nanopore sequencing. This work builds upon a previous SCST application to speech recognition, which used multi-objective loss function combining the standard maximum likelihood loss with a sampled estimate of the expected reward.
We implement an RNN basecaller for nanopore sequencing and train it using the multi-objective SCST loss, which yields a slight improvement in accuracy with respect to a model trained solely with maximum likelihood. We additionally explore the use of SCST loss as a fine-tuning step after maximum likelihood training.

5.2 Introduction

Since the completion of the human genome project in 2003, sequencing technology has become vastly more accessible and affordable. Nanopore sequencing is a new technology that exemplifies this trend: sequencing devices are inexpensive, highly portable, and generate long reads, which facilitates downstream bioinformatics. Despite these advantages, the accuracy of nanopore sequencing lags behind more established methods [53]. Nanopore sequencing works by threading a single strand of DNA through a small pore embedded in a membrane. The DNA alters an electrical current in a sequence-dependent manner, allowing a neural network basecaller to predict each base (A, C, G, or T) as it passes through the pore.

Nanopore basecalling involves segmenting a time series of numeric values with discrete labels, and shares similarities with speech recognition (i.e. segmenting audio frequencies into phonemes or words). Connectionist temporal classification (CTC) [21] trains a recurrent neural network to maximize the probability of the correct sequence labeling, and has seen widespread application in speech recognition, and more recently, to nanopore basecalling [61]. In both of these applications, there is a mismatch between the maximum likelihood (ML) objective used in training, the CTC loss, and the actual metric used for evaluation, the edit distance between the predicted and true labelings. While this metric is discrete and non-differentiable, there has been some success in speech recognition using policy gradient techniques to approximate this gradient and optimize the edit distance directly [20][68]. This work investigates whether similar accuracy gains can be had in nanopore sequencing by adapting these approaches. This project is a novel application of reinforcement learning to bioinformatics and to the field of nanopore sequencing

5.3 Methods

Basecalling with connectionist temporal classification

Under CTC, the output of the neural network defines a probability distribution over possible labelings of the input, in this case the DNA sequence that passed through the pore. CTC uses a differentiable loss function calculated with dynamic programming to calculate the probability of a given labeling. For training, the exact alignment between the input, x, and the true labeling, ℓ^*, is not needed, as dynamic programming is used to marginalize over all alignments and calculate the probability of a label sequence, which depends on the network parameters θ. Using gradient descent, the network is trained to maximize the negative log-likelihood of the correct sequence, $-\log P_\theta(\ell^*|x)$.

Given the input x, the final softmax layer of the neural network outputs y, a $T \times 5$ matrix that specifies the probability of each base plus a gap character at each step of the input. Let $y(t, c)$ be the probability of the character c at time t, where $c \in \{A, C, G, T, \epsilon\}$ and ϵ is a gap (or blank) character representing no change in the pore. Under this model, sequences longer than T are assigned probability 0.

A path π through this output is a length T list of each state, i.e. $\{A, C, G, T, \epsilon\}$. The probability of a given path π is just the product of individual probabilities

$$P_\theta(\pi|y) = \prod_{t=1}^{T} y(\pi_t, t)$$

Each path can be mapped to a label sequence with some function $\mathcal{B}$, which in this case simply removes the gap characters. This is a simplifed version of the canonical CTC model [21] which does not merge repeated labels (so the path `A-CCG--T` would result in the label `"ACCGT"`). For a given label sequence ℓ, we want to find the probability that ℓ was emitted by y, $P_\theta(\ell|y)$. The probability of a label sequence is the sum of all paths that would emit it, essentially marginalizing over all possible positions of gaps:

$$P_\theta(\ell|y) = \sum_{\pi:\mathcal{B}(\pi)=\ell} P_\theta(\pi|x)$$

This probability can be efficiently computed through dynamic programming with a forward algorithm. The forward probability of a label is defined as $\alpha(t, s)$, the probability that the first s characters of ℓ being emitted by time t. The core recursion is,

$$\alpha(t, s) = y(t, \ell_s)\alpha(t-1, s-1) + y(t, \epsilon)\alpha(t-1, s)$$

to which we add the base case of $\alpha(0, 0) = 1$. From this matrix we can easily read out the probability of the full sequence, $P_\theta(\ell|x) = \alpha(T, |\ell|)$.

The task of finding the best label from the output y is referred to as decoding. For basecalling we want to find the best label, $\hat{\ell}$,

$$\hat{\ell} = \max_{\ell} P_\theta(\ell|x)$$

While there is not an efficient general algorithm for this optimization, various heuristic search algorithms can be used to find high probability sequences, such as beam search. In this case, we take a simpler approach is to instead find the single best path:

$$\hat{\pi} = \max_{\pi} P_\theta(\pi|x)$$

Thus, the greedy solution is $\hat{\ell} = \mathcal{B}(\hat{\pi})$. In this model, this is equivalent to taking the argmax of each output in the time series and removing the gaps.

An alternative heuristic search method is beam search, which has been used extensively in the decoding of neural networks, including CTC models [20].

Minimizing errors with reinforcement learning

Given the input x and the true sequence label ℓ^*, the CTC loss is the negative log-likelihood of that label

$$\mathcal{L}_{\text{ML}}(\theta) = -\log P_\theta(\ell^*|x)$$

This likelihood as well as its gradient $\nabla_\theta \mathcal{L}_{\mathrm{ML}}(x, y)$ can be evaluated using dynamic programming, though in this work we make use of the automatic differentiation of CTC loss that is implemented in TensorFlow. Through gradient descent, the objective is to maximize the probability of the true labeling.

The REINFORCE [66] algorithm could be applied to directly optimize the expected reward, minimizing

$$\mathcal{L}_{\mathrm{RL}}(\theta) = -\mathbb{E}_{\ell \sim P_\theta(\ell|x)}[r(\ell)]$$

where the reward of a given label ℓ is defined as

$$r(\ell) = 1 - \mathrm{NormalizedLevenshteinDistance}(\ell, \ell^*)$$

where the Normalized Levenshtein Distance is the Levenshtein edit distance normalized by the length of the true label, ℓ^*.

It is more computationally tractable to sample paths rather than labels directly. As in [20], alignments can be easily sampled from the softmax probabilities output by the network, which are then mapped to labels using the function $\ell^s = \mathcal{B}(\pi_s)$. Using a single sample (as in [68]) the expected reward then simplifies to

$$\mathcal{L}_{\mathrm{RL}}(\theta) = -r(\ell^s) P_\theta(\ell^s|x)$$

As policy gradients can be negatively affected by variance, one remedy is to incorporate a reward baseline that is subtracted from the sampled reward. In self-critical sequence training (SCST)[54], this baseline is the decoded label sequence. Hence, the SCST loss with is

$$\mathcal{L}_{\mathrm{SCST}}(\theta) = -[r(\ell^s) - r(\hat{\ell})] P_\theta(\ell^s|x)$$

In this work we adopt the approach of [68] who introduced a multi-objective loss that incorporates both the maximum likelihood and policy gradient losses,

$$\mathcal{L}(\theta) = \mathcal{L}_{ML}(\theta) + \lambda \mathcal{L}_{SCST}(\theta)$$

When learning this multi-objective policy, λ is a hyperparameter which governs the influence of the SCST on the gradient.

Network architecture

Similar to a previous application of CTC for nanopore basecalling [61], we use a mixed convolutional/recurrent architecture for basecalling. This network, based on our previous basecaller PoreOverNet [59], has a single 1D convolutional layer (filter size of 256, kernel of size 9, stride of 1) followed by three stacked bidirectional GRU layers (with 128 units each). The output is run through a softmax layer and passed to the CTC loss function implemented in TensorFlow.

While a thorough exploration of basecalling network architectures is beyond the scope of this work, recurrent architectures have performed well for nanopore basecalling, and can even outperform newer architectures such as the transformer [48].

Training data

Bonito (`https://github.com/nanoporetech/bonito`), the ONT research basecaller, makes its training set publicly available, which thus serves as a standardized dataset for training nanopore basecallers, much as ImageNet is a standard dataset for image recognition.

The entire dataset comprises 1.2 million input/label pairs, where the input is a window of 3600 signal measurements, and the output is the (variable-length) true reference sequence. The dataset was built from a mix of reads sequenced using R9.4.1 chemistry.

We used a 40/20/40 split on the Bonito training dataset, with 40% of the data used as the training dataset, 20% used as a validation set for early stopping, and 40% of the data held out as a test set for the final evaluation of the model. For both training and evaluation, a batch size of 64 was used. Training was done with the Adam optimizer [31] with a learning rate of 0.001 on a single Nvidia A6000 GPU.

5.4 Results

Multi-objective optimization

The primary goal of this work is testing the multi-objective policy loss function developed in [68] to nanopore basecalling. Training for each architecture was done until the edit distance on the validation set failed to improve for 5 consecutive epochs (the patience hyperparameter). After training, the checkpoint model with the lowest validation error was selected and then evaluated on the held-out test set. A representative training trajectory is shown in Figure 5.1, which depicts the loss decreasing while the reward being optimized by the policy gradient (the accuracy of a sampled label) increases.

Several values of λ, the hyperparameter used to scale the policy gradient loss, were sampled. Additionally, models were trained both with and without the use of the SCST baseline. Baselines are traditionally used in policy gradient approaches to reduce variance of Monte Carlo policy samples, which is even more important in this work which relies on a single sample for each element of the minibatch.

As shown in Table 5.1, these results generally show a small improvement in accuracy from using policy gradient based approaches. A key hyperparameter is λ, which governs the relative weight of the policy gradient loss in the multi-objective loss. With $\lambda = 10$ and no baseline, the performance suffered drastically. In the low λ limit, the multi-objective loss is just the ML loss, so the models with $\lambda = 0.1$ all achieved similar performance. Though the best model was achieved with a baseline and $\lambda = 1$, the performance gain relative to the ML model is still quite small. While these results suggest that, with the right hyperparameters, this reinforcement learning method may outperform a network trained only with maximum likelihood, additional work is needed to establish the significance of these small performance differences.

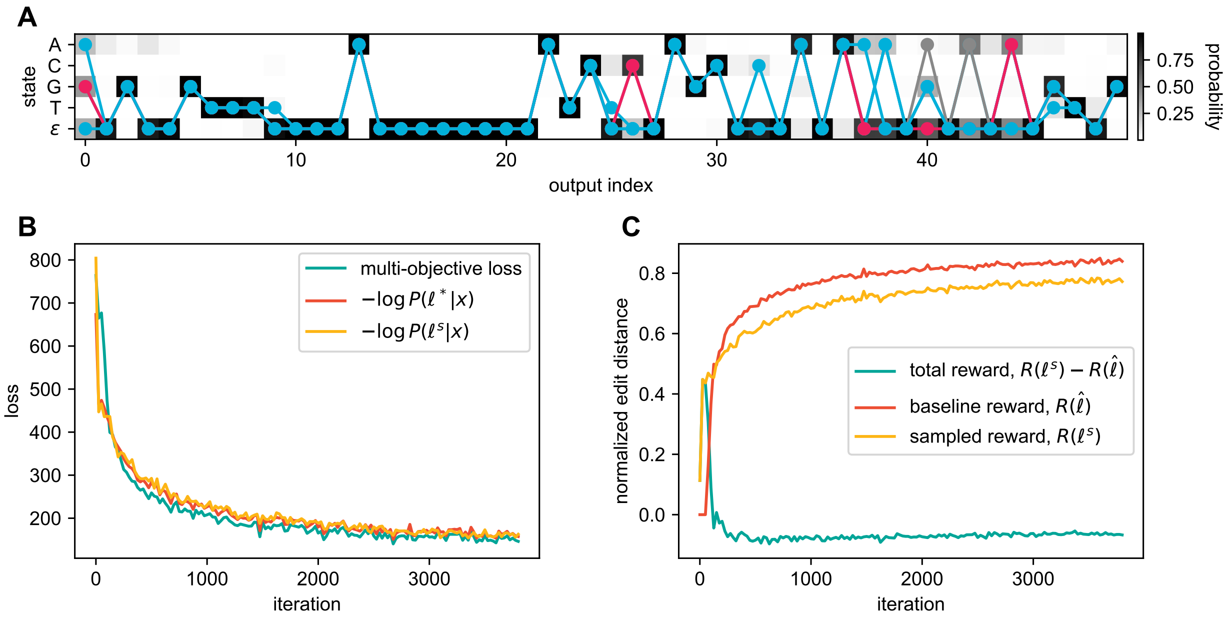


Figure 5.1: Representative example of a basecalling network trained with policy gradient.

(A) The logits output after CTC training define a matrix of probabilities, which correspond to the emission probabilities of a simple profile HMM [58]. Sequences are sampled by paths during the policy gradient and weighted by the reward, in this case the sequence similarity between the decoded sequence and the true label. A baseline from the greedy decoded sequence is subtracted from the sampled reward to yield the total reward. The sampled paths are colored based on reward with baseline (magenta, $r > 0$, gray, $r = 0$, blue $r < 0$). (B) Loss during representative trajectory after a single epoch of training using a multi-objective loss function. At each iteration, the total loss is the sum of the ML loss and the reward-weighted loss from the policy gradient. (C) As with SCST, the sampled reward is subtracted by the baseline reward from the greedy decoding of the sequence. For most sampled paths, the baseline is better than the sampled reward, which yields a net negative reward.

Fine-tuning after maximum likelihood

As has been previously observed [20][68], it is difficult to train a network for speech recognition using solely reinforcement learning, as most samples from randomly initialized networks will be too far from the true label to learn effectively. While [68] circumvented this with the multi-objective loss explored in the previous section, another approach, as adopted in [20], would be to first train a network using maximum likelihood and then fine-tune it with reinforcement learning. Adapting both of these methods, a single benchmark was run by

Run	Normalized edit distance (%)
ML-only	10.29
ML+SCST, no baseline, $\lambda = 0.1$	10.01
ML+SCST, no baseline, $\lambda = 1$	10.01
ML+SCST, no baseline, $\lambda = 10$	15.05
ML+SCST, baseline, $\lambda = 0.1$	10.15
ML+SCST, baseline, $\lambda = 1$	**9.97**
ML+SCST, baseline, $\lambda = 10$	10.29
ML (5 epochs) then ML+SCST, baseline, $\lambda = 1$	**9.83**

Table 5.1: Error rates of different training regimes.

Accuracy assessed on held-out validation test set (lowest values are bolded). ML refers to the maximum likelihood loss, while ML+SCST is a multi-objective loss incoporating both ML and the SCST loss, which is scaled by the hyperparameter λ. Baseline refers to the use of the SCST baseline in policy gradient samples.

training a model with maximum likelihood for 5 epochs and then training with the multi-objective SCST loss. As in previous runs, training was stopped once performance plateaued. The result was a validation error below any of the other models explored, suggesting the promise of a fine-tuning approach.

5.5 Discussion

This work is a novel application of policy gradient techniques to a rapidly developing field of bioinformatics. Using an objective that combines maximum likelihood with an estimate of the expected reward, these results suggest it may be possible to train a network that exceeds the accuracy of one trained solely with maximum likelihood. Though the magnitude of improvement was relatively minor, these results support further exploration of reinforcement learning in this domain.

Though this work used edit distance as the reward function, more complex rewards that do not weight all errors equally could potentially yield larger improvements. For example, homopolymers are a common source of error in nanopore sequencing [53], so a reward function could be developed which more heavily weights errors in homopolymer regions.

Though in this work a fixed λ was used for the entire duration of the simulation, in [68] they switched values of λ when the test error plateaued, which could achieve more consistent results. Alternative sampling-based approaches that also attempt to maximize a non-differentiable reward [43] could also be explored in further work. Finally, while the core focus of this work was on the training procedure, other factors such as the network architecture could also yield improvements in basecalling accuracy.

5.6 Acknowledgments

This work was supported by NIH/NCI grant CA220441, Oxford Nanopore Technologies, and the NVIDIA Academic Hardware Grant Program.

Bibliography

[1] Gunjan Baid et al. "DeepConsensus : Gap-Aware Sequence Transformers for Sequence Correction". In: (2021). DOI: 10.1038/s41587-022-01435-7.

[2] Lukas Biewald. *Experiment Tracking with Weights and Biases*. 2020. URL: https://www.wandb.com/.

[3] Vladimír Boža, Broňa Brejová, and Tomáš Vinař. "DeepNano: Deep recurrent neural networks for base calling in MinION nanopore reads". In: *PLOS ONE* 12.6 (2017), pp. 1–13. DOI: 10.1371/journal.pone.0178751. URL: https://doi.org/10.1371/journal.pone.0178751.

[4] Vladimír Boža et al. "DeepNano-blitz: a fast base caller for MinION nanopore sequencers". In: *Bioinformatics (Oxford, England)* 36.14 (2020), pp. 4191–4192. ISSN: 13674811. DOI: 10.1093/bioinformatics/btaa297.

[5] Clive G. Brown and James Clarke. "Nanopore development at Oxford Nanopore". In: *Nature Biotechnology* 34.8 (2016), pp. 810–811. ISSN: 15461696. DOI: 10.1038/nbt.3622.

[6] Sarah L. Castro-Wallace et al. "Nanopore DNA Sequencing and Genome Assembly on the International Space Station". In: *Scientific Reports* 7.1 (2017), pp. 1–12. ISSN: 20452322. DOI: 10.1038/s41598-017-18364-0. URL: http://dx.doi.org/10.1038/s41598-017-18364-0.

[7] Rachel S.L. Chan, Paul Gordon, and Michael R. Smith. "Evaluation of Dynamic Time Warp Barycenter Averaging (DBA) for its Potential in Generating a Consensus Nanopore Signal for Genetic and Epigenetic Sequences". In: *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS* 2018-July (2018), pp. 2821–2824. ISSN: 1557170X. DOI: 10.1109/EMBC.2018.8512873.

[8] Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014).

[9] Kyunghyun Cho et al. "On the properties of neural machine translation: Encoder–decoder approaches". In: *Proceedings of SSST 2014 - 8th Workshop on Syntax, Semantics and Structure in Statistical Translation* (2014), pp. 103–111. DOI: 10.3115/v1/w14-4012. arXiv: 1409.1259.

[10] Junyoung Chung et al. “Empirical evaluation of gated recurrent neural networks on sequence modeling”. In: *arXiv preprint arXiv:1412.3555* 2015-Septe (2014), pp. 119–124. ISSN: 19457901. arXiv: `1412.3555v1`.

[11] Francis S. Collins, Michael Morgan, and Aristides Patrinos. “The Human Genome Project: Lessons from large-scale biology”. In: *Science* 300.5617 (2003), pp. 286–290. ISSN: 00368075. DOI: `10.1126/science.1084564`.

[12] Matei David et al. “Nanocall: an open source basecaller for Oxford Nanopore sequencing data”. In: *Bioinformatics* 33.1 (2016), pp. 49–55.

[13] David Deamer, Mark Akeson, and Daniel Branton. “Three decades of nanopore sequencing”. In: *Nature Biotechnology* 34.5 (2016), p. 518.

[14] David W. Deamer and Mark Akeson. “Nanopores and nucleic acids: Prospects for ultrarapid sequencing”. In: *Trends in Biotechnology* 18.4 (2000), pp. 147–151. ISSN: 01677799. DOI: `10.1016/S0167-7799(00)01426-8`.

[15] Ian M Derrington et al. “Nanopore DNA sequencing with MspA”. In: *Proceedings of the National Academy of Sciences* 107.37 (2010), pp. 16060–16065.

[16] Richard Durbin et al. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press, 1998.

[17] Robert C. Edgar. “Search and clustering orders of magnitude faster than BLAST”. In: *Bioinformatics* 26.19 (2010), pp. 2460–2461. ISSN: 13674803. DOI: `10.1093/bioinformatics/btq461`.

[18] Nuno Rodrigues Faria et al. “Mobile real-time surveillance of Zika virus in Brazil”. In: *Genome Medicine* 8.1 (2016), pp. 2–5. ISSN: 1756994X. DOI: `10.1186/s13073-016-0356-2`. URL: `http://dx.doi.org/10.1186/s13073-016-0356-2`.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[20] Alex Graves and Navdeep Jaitly. “Towards end-to-end speech recognition with recurrent neural networks”. In: *International Conference on Machine Learning*. 2014, pp. 1764–1772.

[21] Alex Graves et al. “Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks”. In: *Proceedings of the 23rd International Conference on Machine Learning*. ICML ’06. ACM. New York, NY, USA: ACM, 2006, pp. 369–376. ISBN: 1-59593-383-2. DOI: `10.1145/1143844.1143891`. URL: `http://doi.acm.org/10.1145/1143844.1143891`.

[22] James M. Heather and Benjamin Chain. “The sequence of sequencers: The history of sequencing DNA”. In: *Genomics* 107.1 (2016), pp. 1–8. ISSN: 10898646. DOI: `10.1016/j.ygeno.2015.11.003`. URL: `http://dx.doi.org/10.1016/j.ygeno.2015.11.003`.

[23] Sepp Hochreiter et al. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780. DOI: `10.3138/9781487583064-002`.

[24] Ian Holmes and Richard Durbin. "Dynamic programming alignment accuracy". In: *Journal of computational biology* 5.3 (1998), pp. 493–504.

[25] Neng Huang et al. "NeuralPolish: a novel Nanopore polishing method based on alignment matrix construction and orthogonal Bi-GRU Networks". In: *Bioinformatics* (2021), pp. 1–8. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btab354.

[26] Miten Jain et al. "Nanopore sequencing and assembly of a human genome with ultra-long reads". In: *Nature Biotechnology* 36 (Jan. 2018), p. 338. URL: https://doi.org/10.1038/nbt.4060%20http://10.0.4.14/nbt.4060%20https://www.nature.com/articles/nbt.4060%7B%5C#%7Dsupplementary-information.

[27] Miten Jain et al. "The Oxford Nanopore MinION: delivery of nanopore sequencing to the genomics community". In: *Genome Biology* 17.1 (2016), pp. 1–11. ISSN: 1474-760X. DOI: 10.1186/s13059-016-1103-0. URL: http://dx.doi.org/10.1186/s13059-016-1103-0.

[28] Sarah S. Johnson et al. "Real-time DNA sequencing in the antarctic dry valleys using the Oxford nanopore sequencer". In: *Journal of Biomolecular Techniques* 28.1 (2017), pp. 2–7. ISSN: 19434731. DOI: 10.7171/jbt.17-2801-009.

[29] John Jumper et al. "Highly accurate protein structure prediction with AlphaFold." In: *Nature* (2021). ISSN: 1476-4687. DOI: 10.1038/s41586-021-03819-2. URL: http://www.ncbi.nlm.nih.gov/pubmed/34265844.

[30] Søren M. Karst et al. "High-accuracy long-read amplicon sequences using unique molecular identifiers with Nanopore or PacBio sequencing". In: *Nature Methods* (2021). ISSN: 15487105. DOI: 10.1038/s41592-020-01041-y.

[31] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: (2014), pp. 1–15. arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980.

[32] Mikhail Kolmogorov et al. "Assembly of long, error-prone reads using repeat graphs". In: *Nature biotechnology* 37.5 (2019), pp. 540–546.

[33] Samuel Kriman et al. "QuartzNet: Deep Automatic Speech Recognition with 1D Time-Channel Separable Convolutions". In: (2019), pp. 2–6. arXiv: 1910.10261. URL: http://arxiv.org/abs/1910.10261.

[34] John Lafferty, Andrew McCallum, and Fernando C N Pereira. "Conditional random fields: Probabilistic models for segmenting and labeling sequence data". In: (2001).

[35] Andrew H. Laszlo et al. "Decoding long nanopore sequencing reads of natural DNA". In: *Nature Biotechnology* 32.8 (2014), pp. 829–833. ISSN: 15461696. DOI: 10.1038/nbt.2950.

[36] Y LeCun, Y Bengio, and G Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444.

[37] Heng Li. “Minimap and miniasm: Fast mapping and de novo assembly for noisy long sequences”. In: *Bioinformatics* 32.14 (2016), pp. 2103–2110. ISSN: 14602059. DOI: 10.1093/bioinformatics/btw152. arXiv: 1512.01801.

[38] Nicholas J Loman, Joshua Quick, and Jared T Simpson. “A complete bacterial genome assembled de novo using only nanopore sequencing data”. In: *Nature Methods* 12.8 (2015), p. 733.

[39] Romain Lopez et al. “Deep generative modeling for single-cell transcriptomics”. In: *Nature Methods* 15.12 (2018), pp. 1053–1058. ISSN: 15487105. DOI: 10.1038/s41592-018-0229-2. URL: http://dx.doi.org/10.1038/s41592-018-0229-2.

[40] Santiago Marco-Sola et al. “Fast gap-affine pairwise alignment using the wavefront algorithm”. In: *Bioinformatics* (2020), pp. 1–8. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btaa777.

[41] Michael L Metzker. “Sequencing technologies — the next generation”. In: *Nature Reviews Genetics* 11 (Dec. 2010), p. 31. URL: https://doi.org/10.1038/nrg2626%20http://10.0.4.14/nrg2626.

[42] Samuel M Nicholls et al. “Ultra-deep, long-read nanopore sequencing of mock microbial community standards”. In: *GigaScience* 8.5 (2019). ISSN: 2047-217X. DOI: 10.1093/gigascience/giz043. URL: https://doi.org/10.1093/gigascience/giz043.

[43] Mohammad Norouzi et al. “Reward Augmented Maximum Likelihood for Neural Structured Prediction”. In: *Advances in Neural Information Processing Systems 29*. Ed. by D D Lee et al. Curran Associates, Inc., 2016, pp. 1723–1731. URL: http://papers.nips.cc/paper/6547-reward-augmented-maximum-likelihood-for-neural-structured-prediction.pdf.

[44] Sergey Nurk et al. “The complete sequence of a human genome”. In: *Science* 376.6588 (2022), pp. 44–53.

[45] Oxford Nanopore Technologies. *Bonito*. URL: https://github.com/nanoporetech/bonito.

[46] Oxford Nanopore Technologies. *Medaka*. URL: https://github.com/nanoporetech/medaka.

[47] Oxford Nanopore Technologies. *Remora*. URL: https://github.com/nanoporetech/remora.

[48] Marc Pages-Gallego and Jeroen de Ridder. “Comprehensive and standardized benchmarking of deep learning architectures for basecalling nanopore sequencing data”. In: *bioRxiv* (2022), p. 2022.05.17.492272. URL: https://www.biorxiv.org/content/10.1101/2022.05.17.492272v1%7B%5C%%7D0Ahttps://www.biorxiv.org/content/10.1101/2022.05.17.492272v1.abstract.

[49] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “On the difficulty of training recurrent neural networks”. In: *International Conference on Machine Learning*. 2013, pp. 1310–1318.

[50] Aaron Pomerantz et al. “Real-time DNA barcoding in a rainforest using nanopore sequencing: Opportunities for rapid biodiversity assessments and local capacity building”. In: *GigaScience* 7.4 (2018), pp. 1–14. ISSN: 2047217X. DOI: `10.1093/gigascience/giy033`.

[51] Ploy N. Pratanwanich et al. “Identification of differential RNA modifications from nanopore direct RNA sequencing with xPore”. In: *Nature Biotechnology* 39.11 (2021), pp. 1394–1402. ISSN: 15461696. DOI: `10.1038/s41587-021-00949-w`.

[52] Arthur C. Rand et al. “Mapping DNA methylation with high-throughput nanopore sequencing”. In: *Nature Methods* 14.4 (2017), pp. 411–413. ISSN: 15487105. DOI: `10.1038/nmeth.4189`.

[53] Franka J Rang, Wigard P Kloosterman, and Jeroen de Ridder. “From squiggle to basepair: computational approaches for improving nanopore sequencing read accuracy”. In: *Genome Biology* 19.1 (2018), p. 90.

[54] Steven J Rennie et al. “Self-critical sequence training for image captioning”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 7008–7024.

[55] Mantas Sereika et al. “Oxford Nanopore R10.4 long-read sequencing enables the generation of near-finished bacterial genomes from pure cultures and metagenomes without short-read or reference polishing”. In: *Nature Methods* 19.7 (2022), pp. 823–826. ISSN: 15487105. DOI: `10.1038/s41592-022-01539-7`.

[56] Kishwar Shafin et al. “Efficient de novo assembly of eleven human genomes using PromethION sequencing and a novel nanopore toolkit”. In: *bioRxiv* (2019). DOI: `10.1101/715722`. URL: `https://www.biorxiv.org/content/early/2019/07/26/715722`.

[57] Kishwar Shafin et al. “Haplotype-aware variant calling with PEPPER-Margin-DeepVariant enables high accuracy in nanopore long-reads”. In: *Nature Methods* 18.11 (2021), pp. 1322–1332. ISSN: 15487105. DOI: `10.1038/s41592-021-01299-w`.

[58] Jordi Silvestre-Ryan and Ian Holmes. “Consensus Decoding of Recurrent Neural Network Basecallers”. In: *Algorithms for Computational Biology*. Ed. by Jesper Jansson, Carlos Martín-Vide, and Miguel A Vega-Rodríguez. Cham: Springer International Publishing, 2018, pp. 128–139. ISBN: 978-3-319-91938-6.

[59] Jordi Silvestre-Ryan and Ian Holmes. “Pair consensus decoding improves accuracy of neural network basecallers for nanopore sequencing”. In: *Genome Biology* 22.1 (2021), p. 38. ISSN: 1474-760X. DOI: `10.1186/s13059-020-02255-1`. URL: `https://doi.org/10.1186/s13059-020-02255-1`.

[60] Marcus Stoiber and James Brown. "BasecRAWller: Streaming Nanopore Basecalling Directly from Raw Signal". In: *bioRxiv* (2017), p. 133058.

[61] Haotian Teng et al. "Chiron: translating nanopore raw signal directly into nucleotide sequence using deep learning". In: *GigaScience* 7.5 (2018), giy037. DOI: `10.1093/gigascience/giy037`. URL: `http://dx.doi.org/10.1093/gigascience/giy037`.

[62] Robert Vaser et al. "Fast and accurate de novo genome assembly from long uncorrected reads". In: *Genome Research* 27.5 (2017), pp. 737–746. ISSN: 15495469. DOI: `10.1101/gr.214270.116`.

[63] Ashish Vaswani et al. "Attention is all you need". In: *Advances in Neural Information Processing Systems*. 2017, pp. 5998–6008.

[64] Roger Volden et al. "Improving nanopore read accuracy with the R2C2 method enables the sequencing of highly multiplexed full-length single-cell cDNA". In: 115.39 (2018), pp. 1–6. DOI: `10.1073/pnas.1806447115`.

[65] Ryan R. Wick, Louise M. Judd, and Kathryn E. Holt. "Performance of neural network basecalling tools for Oxford Nanopore sequencing". In: *Genome Biology* 20.1 (Jan. 2019), p. 129. ISSN: 1474-760X. DOI: `10.1186/s13059-019-1727-y`. URL: `https://genomebiology.biomedcentral.com/articles/10.1186/s13059-019-1727-y`.

[66] Ronald J. Willia. "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: *Machine Learning* 8.3 (1992), pp. 229–256. ISSN: 15730565. DOI: `10.1023/A:1022672621406`.

[67] Yonghui Wu et al. "Google's neural machine translation system: Bridging the gap between human and machine translation". In: *arXiv preprint arXiv:1609.08144* (2016).

[68] Yingbo Zhou, Caiming Xiong, and Richard Socher. "Improving end-to-end speech recognition with policy learning". In: *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings* 2018-April (2018), pp. 5819–5823. ISSN: 15206149. DOI: `10.1109/ICASSP.2018.8462361`.