PERGAMON

Available at
www.ElsevierMathematics.com
POWERED BY SCIENCE DIRECT®

Journal of the Franklin Institute 341 (2004) 23–36

Journal of The Franklin Institute

www.elsevier.com/locate/jfranklin

Basecalling using hidden Markov models

Petros Boufounos[a,*,1], Sameh El-Difrawy[b], Dan Ehrlich[b]

[a] *Research Lab For Electronics, Massachussetts Institute of Technology, 77 Massachussetts Avenue, Room 36-631, Cambridge, MA 02139, USA*

[b] *Whitehead Institute for Biomedical Research, 9 Cambridge Center, Cambridge, MA 02142, USA*

Abstract

In this paper we propose hidden Markov models to model electropherograms from DNA sequencing equipment and perform basecalling. We model the state emission densities using artificial neural networks, and modify the Baum–Welch reestimation procedure to perform training. Moreover, we develop a method that exploits consensus sequences to label training data, thus minimizing the need for hand labeling. We propose the same method for locating an electropherogram in a longer DNA sequence. We also perform a careful study of the basecalling errors and propose alternative HMM topologies that might further improve performance. Our results demonstrate the potential of these models. Based on these results, we conclude by suggesting further research directions.

Keywords: Hidden Markov models; Basecalling; DNA sequencing; PHRED

1. Introduction and background

In recent years DNA sequencing has become a popular tool in Biology, significantly affecting the practice in the field. The impact of this method has created a need to automate the translation of sequencing signals (*electropherograms*) to the corresponding sequence of bases, a process known as *basecalling*.

If we skip the details on the chemistry, data generation, data collection and preprocessing, basecalling is a simple problem to describe, but not easy to solve. Indeed, basecalling is the process of converting a time signal, such as the one in Fig. 1, to a string of A, T, C, and G—AAACCCCCTGAATATG in this particular

*Corresponding author. Tel.: +1-617-258-5819; fax: +1-617-253-8495.
E-mail addresses: petrosb@mit.edu (P. Boufounos), sameh@wi.mit.edu (S. El-Difrawy), ehrlich@wi.mit.edu (D. Ehrlich).

[1] This author performed the work while at Whitehead Institute.

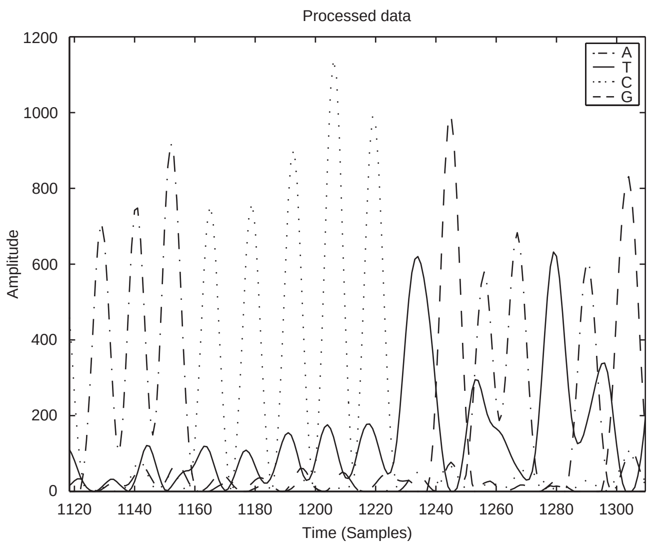


Fig. 1. A sample electropherogram. Basecalling is the task of converting this time signal to a sequence of letters—AAACCCCCTGAATATG in this case.

example. Unfortunately, signals are not often as clean. Crosstalk is significant between the four channels, and peaks merge, especially near the end of the electropherogram. Other artifacts might show up, such as the ambiguity in the ATG call near the end of the example in the figure. The task is probabilistic in nature, so a good basecaller should also provide some measure of the quality of the read. These measures are especially significant in genome sequencing projects since they are used in subsequent genome assembly steps, such as [1,2].

The most successful basecaller is PHRED [3,4], currently used by the Human Genome project. More recently researchers have attempted to provide some statistical foundations to the problem and use statistical models to solve it (see e.g. [5–7]). In fact, in [6] the process is modeled as a Markov chain and analyzed using Markov chain Monte Carlo methods.

In this paper we use hidden Markov models (HMMs) to provide an alternative statistical description to basecalling. Our approach has the advantage that it does not assume a particular peak shape at the cost of requiring some initial training. Since generating training data can be labor intensive, especially for new equipment, we describe a quick method to do so. This method is generally useful in instances where electropherograms need to be located in a larger sequence. This paper expands on the work described in [8].

2. Motivation and theoretical background

A key observation is that the DNA basecalling problem is similar to the speech recognition problem: a time signal should be translated to a sequence of symbols under a particular set of rules. This similarity makes HMMs—widely used in speech recognition—potentially applicable to basecalling. The main difference is that in speech recognition problems the grammar of a spoken language forms the rules. In DNA sequencing problems the rules are simple: the sequence is an i.i.d. process drawn from {A, T, C, G}.

While an excellent tutorial on HMMs, is contained in [9], we provide a brief overview here. This overview also establishes the notation used in the rest of the paper. Furthermore, we introduce the combination of HMMs with neural networks used in our implementation.

2.1. Markov chains and hidden Markov models

A Markov chain is a discrete random process $q[t], t = 1, \dots, N$ with the property that $P(q[t] \| q[t-1], q[t-2], \dots, q[1]) = P(q[t] \| q[t-1])$. In other words, at any time $q[t]$ incorporates all the information about the past and is called the *state* of the process. We assume that $q[t]$ belongs in a discrete and finite set—the state space of the process. To describe such a process, we need to specify the initial probabilities of each state, and the transition probabilities from state to state:

$$\pi_i = P(q[1] = i), \tag{1}$$

$$a_{ij} = P(q[t] = i | q[t-1] = j) \quad \forall t > 1. \tag{2}$$

The Markov chain becomes hidden if we assume that the states are not observed directly, but through some probabilistic output vector $\mathbf{O}[t]$ that is independent of everything else conditional on the current state. The distribution of the observation vector—also called *emission probability*—only depends on the current state. Thus, for any time t there is a corresponding probability of each state given the observation at that time:[2]

$$b_i[t] = P(q[t] = i | \mathbf{O}[t]) \tag{3}$$

$$= \frac{P(\mathbf{O}[t] \| q[t] = i) \cdot P_i}{P(\mathbf{O}[t])}, \tag{4}$$

where $P_i = P(q[t] = i)$ is the unconditional probability of being at state i at any time. This formulation requires that $b_i[t]$ is a probability mass function that sums to 1 over all i. On the other hand, the more conventional approach described in [9] defines $b_i[t] = P(\mathbf{O}[t] \| q[t] = i)$, making it a probability density function that integrates to 1 over all possible $\mathbf{O}[t]$.

We denote the whole model by λ, and use it to compute $\alpha_i[t] = [P(\mathbf{O}[1 \dots t], q[t] = i | \lambda)]/[P(\mathbf{O}[1 \dots t])]$ and $\beta_i[t] = [P(\mathbf{O}[(t+1) \dots T] \| q[t] = i, \lambda)]/[P(\mathbf{O}[(t+1) \dots T])]$, also known as the forward and the backward variables, respectively. We can compute

[2] Note that this formulation is different from the usual one, such as in [9]. The differences propagate in the subsequent algorithms. This formulation allows us to accommodate artificial neural networks (ANNs) for the emissions model in Section 2.2. More details on the modifications can be found in [10,11]

these using the forward–backward algorithm:

$$\alpha_j[t] = \begin{cases} \pi_j \dfrac{b_j[1]}{P_j}, & t = 1, \\ \left[\sum_{i=1}^{N} \alpha_i[t-1]a_{ij}\right] \dfrac{b_j[t]}{P_j}, & t > 1, \end{cases} \tag{5}$$

$$\beta_i[t] = \begin{cases} 1, & t = T, \\ \sum_{j=1}^{N} a_{ij} \dfrac{b_j[t+1]}{P_j} \beta_j[t+1], & t < T. \end{cases} \tag{6}$$

Finally, we can estimate $\gamma_i[t] = P(q[t] = i|\mathbf{O}[1 \dots T], \lambda)$ and $\xi_{ij}[t] = P(q[t] = i, q[t+1] = j|\mathbf{O}[1 \dots T], \lambda)$ using

$$\xi_{ij}[t] = \frac{\alpha_i[t]a_{ij}(b_j[t+1]/P_j)\beta_j[t+1]}{\sum_{k=1}^{N}\sum_{l=1}^{N} \alpha_k[t]a_{kl}(b_l[t+1]/P_l)\beta_l[t+1]}, \tag{7}$$

$$\gamma_i[t] = \frac{\alpha_i[t]\beta_i[t]}{\sum_{j=1}^{N} \alpha_j[t]\beta_j[t]} = \sum_{j=1}^{N} \xi_{ij}[t]. \tag{8}$$

To model state emission densities, Gaussian mixture models provide fast training and have been extensively studied. However, early experimentation with the data showed that ANNs can capture the state emission statistics more accurately given our data. Although better feature selection might improve the performance of Mixture models, this is beyond the scope of our work.

The network we use is a typical feedforward layered network with sigmoid activation functions. That means that the output y_i of each node of each layer is $y_i = f\left(w_0 + \sum_j w_j x_j\right)$, where $f(x) = (1 + \mathrm{e}^{-x})^{-1}$, x_j are the inputs, and w_j are the weights to be estimated. To guarantee that the network produces a distribution function, we use a final "softmax" layer such that, at the output, $y_i = \mathrm{e}^{\alpha x_i}/\sum_j \mathrm{e}^{\alpha x_j}$—where α is a training parameter, independent of i. The sum of the outputs of this layer is always equal to 1, as desired. Had we used the formulation in [9] instead, we would need to configure the network to integrate to 1 over all possible range of inputs, a constraint not as simple to implement as adding a softmax layer at the output.

2.2. Training the model

To train the overall model, we modify the Baum–Welch reestimation algorithm—an instance of the Expectation–Maximization method. The forward–backward algorithm, as described above, performs the Expectation step.

The Maximization step uses γ and ξ to estimate the model parameters:

$$\bar{\pi}_i = \gamma_i[1], \tag{9}$$

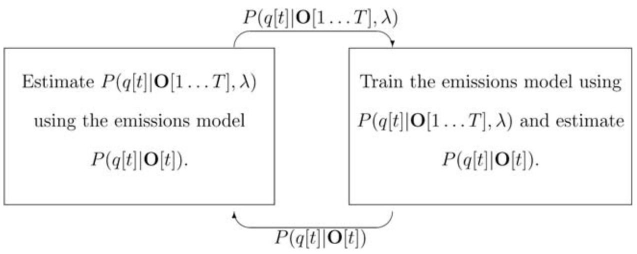

$P(q[t]|\mathbf{O}[1\ldots T],\lambda)$
Estimate $P(q[t]|\mathbf{O}[1\ldots T],\lambda)$ using the emissions model $P(q[t]|\mathbf{O}[t])$.
Train the emissions model using $P(q[t]|\mathbf{O}[1\ldots T],\lambda)$ and estimate $P(q[t]|\mathbf{O}[t])$.
$P(q[t]|\mathbf{O}[t])$

Fig. 2. The modified Baum–Welch procedure.

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_{ij}[t]}{\sum_{t=1}^{T-1} \gamma_j[t]}, \tag{10}$$

$$\bar{P}_i = \frac{\sum_{t=1}^{T} \gamma_i[t]}{T}. \tag{11}$$

At this step, we also need to update the emissions model, so we use $(\mathbf{O}[t], \gamma_i[t])$ as input–output vector pairs to train the ANN using the backpropagation method to perform gradient descent.

The proposed algorithm should be considered as an instance of the generalized EM algorithms. The generalization involves just improving on the lower bound of the cost function at each M-step—through some iterations of the gradient descent—instead of fully maximizing that bound as described in [12].

The training schedule is summarized in Fig. 2. Boufounos [10] and Hennebert et al. [11] provide an extended discussion of the method, including further comments on convergence. Note that the training algorithm preserves the structure of state transition parameters. Indeed, any a_{ij} set to 0 initially will remain 0 after each training iteration.

2.3. Estimating the state transitions

The last problem to solve is how to use a trained model to perform basecalling. As we discuss in the next section, this is equivalent to estimating the most likely state transition path given the data and the model. To do so we define two time variables: $\delta_i[t] = \max_{q[1,\ldots,t-1]} P(q[1,\ldots,t-1], q[t]=i, \mathbf{O}[1,\ldots,t]|\lambda)$, and $\psi_i[t]$ keeps track of the state transitions in δ. These can be computed using the Viterbi algorithm:

$$\delta_j[t] = \begin{cases} \pi_j \dfrac{b_j[1]}{P_j}, & t=1, \\ \left[\max_i \delta_i[t-1]a_{ij}\right](b_j[t]/P_j), & t>1, \end{cases} \tag{12}$$

$$\psi_j[t] = \begin{cases} 0, & t=1, \\ \operatorname{argmax}_i\, [\delta_i[t-1]a_{ij}], & t>1. \end{cases} \tag{13}$$

The most likely state sequence $q^*[t]$ can be back-traced from $\psi_j[t]$ using

$$q^*[t] = \begin{cases} \operatorname{argmax}_i \delta[T], & t = T, \\ \psi_{q^*[t+1]}[t+1], & t < T. \end{cases} \tag{14}$$

The Viterbi algorithm is linear in number of states and observation sequence length, making it very efficient and scalable. These features make the algorithm attractive for basecalling.

A slight variation of the Viterbi algorithm can also be used to generate some training data. The same variation is also useful to locate electropherograms in a longer sequence. We discuss both applications in Section 4.

3. Model selection for DNA sequencing

To implement the system, we need to set the structure of the Markov chain. In this section, we describe a simple model motivated by the shape of the data. Furthermore, we propose alternative configurations, motivated by analyzing the performance of simple model.

3.1. The basic models for recognition and training

We can model DNA sequences using the Markov chain (Fig. 3(a)), which generates an i.i.d. sequence, as desired.[3] In the higher level model, we represent each state by the three-state model of Fig. 3(b), corresponding to the rise, the apex, and the fall of the corresponding peak in the electropherogram. The resulting model is shown in Fig. 3(c). On that model we use the Viterbi algorithm from the previous section to perform basecalling. Indeed, we call a base every time the optimal state transition path passes through a base.

Similarly, we need a model to train the system. This should be generated by the sequences corresponding to the training data. For example the sequence AATCA should produce the model in Fig. 4. We feed this model together with the corresponding electropherogram to the Baum–Welch algorithm to train the system. We use the same type of model in Section 4 to generate training data using a variation of the Viterbi algorithm, and execute database queries straight from the electropherogram. This model contains a number of states linear in the order of the training data, requiring significant computation and memory. However, this model is only used to train the system once, making its size only a minor inconvenience. Furthermore, techniques such as splitting the

[3] It has been noted that in certain DNA regions base sequences are not i.i.d. Instead, certain sequences are more likely to occur than others. However, we choose i.i.d. base distribution to make the model simple and agnostic to the type of region being sequenced. If more information is available for a specific electropherogram, it can be incorporated in the model to improve the results.

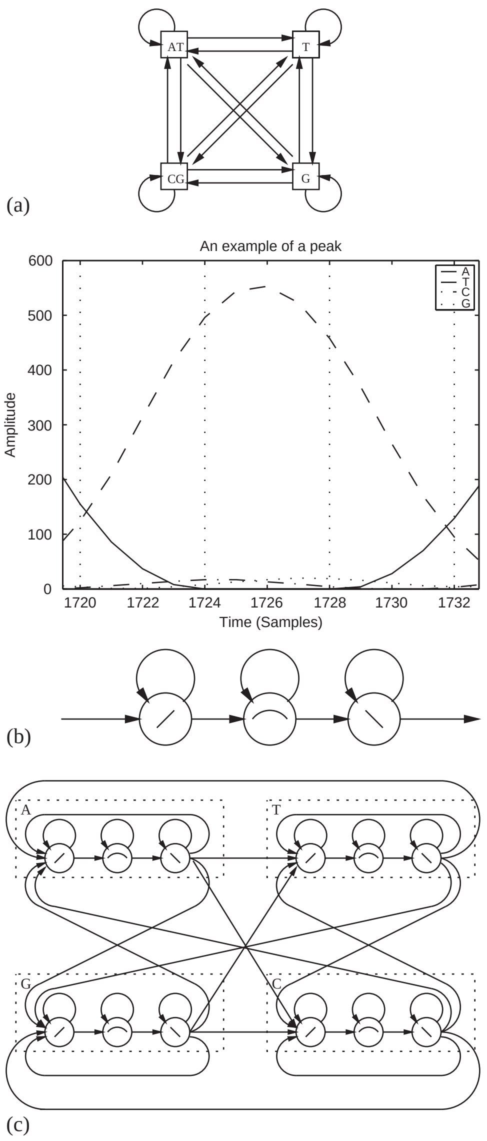


Fig. 3. The model used for basecalling. (a) Model of an i.i.d base sequence (all transitions have probability $\frac{1}{4}$), (b) the model for each base and the part of the peak corresponding to each state, (c) the result of combining (a) and (b).

data in pieces and optimal pruning of branches can be used to reduce the computation.[4]

[4] Such techniques are abundant in the speech processing literature, since the models used for speech recognition are quite complex compared to our model.

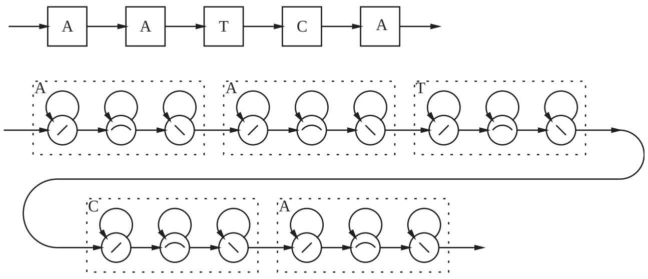


Fig. 4. The left-to-right model used to train the basecaller, generated from the sequence of the training samples.

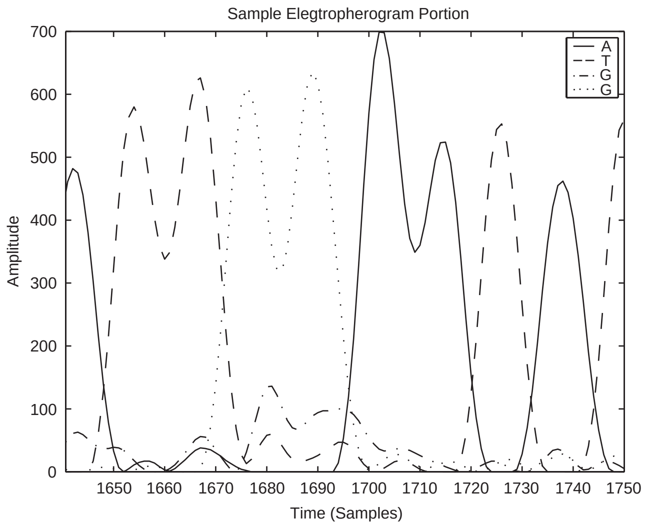
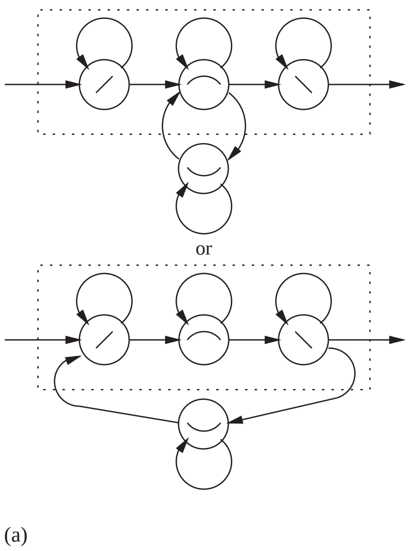


Fig. 5. An example of how model statistics depend on the following Base. Same base transitions, such as A to A create peaks that do not fall all the way to 0 but have a soft dip before rising again, in contrast to transitions to different bases.

3.2. *Alternative model configurations*

The topologies we propose are not the only possible ones, and probably not the best ones. Certain effects are not captured by the model of Fig. 3. For example, as the results verify, the statistics of the fall of the peak vary, depending on what base follows. Fig. 5 shows that transitions from a base to itself, say an A to an A, look very different than transitions from a base to a different one, say a T to a G. The peaks of the later type fall all the way to zero, while the peaks of the former merge. The first peak thus falls to a certain level and a soft dip is created before the second peak rises.

To accommodate such issues we suggest some model modifications Fig. 6. For instance, Fig. 6(a) shows an additional state that can capture the merging of the peaks described above.



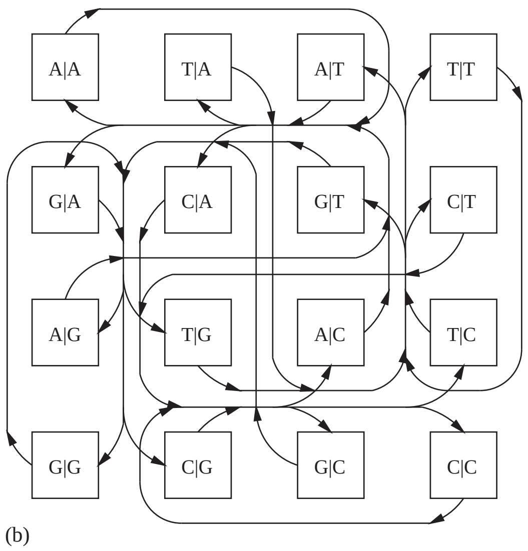


Fig. 6. Alternative models that can be used for basecalling. (a) A replacement for Fig. 3(b) intended to reduce undercalls, (b) a model that incorporates second-order effects.

We can go a step further and assume a second-order underlying Markov chain. This is equivalent to a first-order underlying Markov chain on the last two letters of the sequence instead of only the last one. This 16-state chain will replace the one in Fig. 3(a) with the one in Fig. 6(b). Similar to the simple model, each of the boxes in the figure corresponds to a three-state sequence, as in Fig. 3(b), and the result is a 48-state HMM.

This model captures any effects that require a second-order Markov chain, such as the merging of the peaks described above and the compression effects described in [13]. For example, GC compressions make C peaks arrive earlier when a G peak is preceding. The second-order model will capture this effect in the transition probabilities and the observation density corresponding to the C|G state. Further discussion on alternative models can be found in [10].

4. Generating training data and model training

One issue we encountered was the lack of labeled training data, especially for new equipment. Hand labeling of electropherograms is very expensive and labor intensive. Basecaller-labeled data, on the other hand, are inaccurate for training purposes and create a chicken-and-egg problem. A useful by-product is a method to locate electropherograms in longer sequences. This can be useful to execute queries straight from the electropherogram into large sequence databases.

Instead of the labeled data, however, we can use electropherograms of fragments of published consensus sequences. Yet, these preprocessed electropherograms might not all start from the same location in the sequence and might not all have the same usable length. In order to generate labeled data, we need to find which portion of the sequence corresponds to each electropherogram.

To locate electropherograms in the consensus sequences we use a variation of the Viterbi algorithm. We apply it on a left-to-right model like the one in Fig. 4, generated by the consensus sequence. This model is large and requires significant computation, but we only need to label the data once. In any case, Moore's law makes this method cheaper and faster than hand-labeling.

The data generation involves partially training a HMM, with a set of parameters denoted by λ_p, using very few labeled electropherograms. To obtain these, some minimal human labor is necessary, either to locate them in the consensus sequence, or to hand-label them. Based on λ_p we generate the left-to-right model with parameters λ_c which corresponds to the consensus sequence. The difference is that we assume that the initial probabilities are the same for all the states: $\pi_i = 1/3N$, where N is the number of bases in the consensus sequence. Then we execute the Viterbi algorithm on the electropherograms that we need to locate in the consensus sequence. Essentially, we instruct the model to treat all the bases in the consensus sequence as equally likely starting points of the electropherogram, and find where the data fit best. Indeed, this results to the most likely path in the sequence, which we use to label the data. The method we propose only requires a poorly trained model because it exploits the significant side information embedded in the consensus sequence.

With a fully trained model instead of a partially trained one, this method can also be used to execute queries. Specifically, one could use the model to locate an electropherogram in a longer sequence, without going through the basecalling step and comparing the text sequences. In certain cases—such as low-quality reads—this might result to higher accuracy in the query results. Details of the process are described in [10].

5. Implementation and results

We implemented the proposed simple model in MATLAB and evaluated its performance on 10 electropherograms of PBluescript sequences (different than the training sequences), preprocessed with the standard software of the ABI 377 sequencing system with primer-dye chemistry. In this section we describe the implementation details, and discuss the results.

5.1. Implementation

The input feature vector we use is a 33-sample window of the four-channel electropherogram, centered at the current time point. The 132 points of the feature vector are normalized to have a maximum of 1. An electropherogram decays slowly and exponentially in height, so we use the normalization to make the features consistent during the whole run.

The neural network we use has three hidden sigmoid layers of size 120, 60, and 12 nodes from the input to the output, respectively. We chose three instead of the two hidden layers usually used because the output layer is a softmax function that only has one training parameter.

5.2. Evaluation and results

To measure the accuracy of basecallers there are three types of errors that should be taken into account. *Insertions* occur if a base is called where nothing should have been called. For example, a basecaller performs an insertion error if it calls AATCG while it should have called AACG. Similarly, *deletions* occur if a base is not called when it should have been. For example, a deletion error occurs if the true sequence is AATCG but AACG is called. Finally, *substitution* errors occur if the basecaller calls a different base for a specific location. An example is calling AATCG instead of ATTCG.

The read quality deteriorates as the read length increases. Therefore, reporting error rates at different read length for different systems provides no basis for comparison. It is important to report these errors as a function of read length. In fact, it can be misleading since at small read lengths most systems are essentially error-free.

To evaluate our system we compare our calls to the consensus sequence [14] using CROSS_MATCH [3] to determine the number of insertion, deletion and substitution errors. Fig. 7 shows the results. For comparison, we provide the performance of PHRED version 0.990722g on the same data, over the same region of the electropherograms. We are plotting the average number of errors of each type versus the read length, as well as the average of the total number of errors. We should note here that PHRED has been heavily optimized over the years. Furthermore, it uses its own preprocessor, tuned to work well with the basecaller. On the other hand, we should acknowledge that we are using a limited sample for a thorough comparison. Still, the results provide a guidance about the potential of this method, especially if the modifications described in Sections 3.2 and 6 are implemented.

From the results we see that although the HMM basecaller performs better than PHRED in terms of insertions and substitutions, it generates a significant number of deletion errors. However, the overall performance is comparable, encouraging further research and fine tuning.

In fact, close inspection of the deletion errors validates our suggestions in Section 3.2. As expected, the basecaller often merges bases of the same type and recognizes them as only one peak. In other words our basecaller cannot tell apart two subsequent peaks of the same base near the end of electropherogram that diffuse and merge to form a double peak. For example it would call a GAATC as GATC. Thus, the models in Fig. 6 should significantly improve performance. Indeed this model should be able to incorporate the transition, for example, from an A peak to an A peak in the statistics of the A|A state, thus reducing the number of such deletions—which are the dominant type of error in the system.

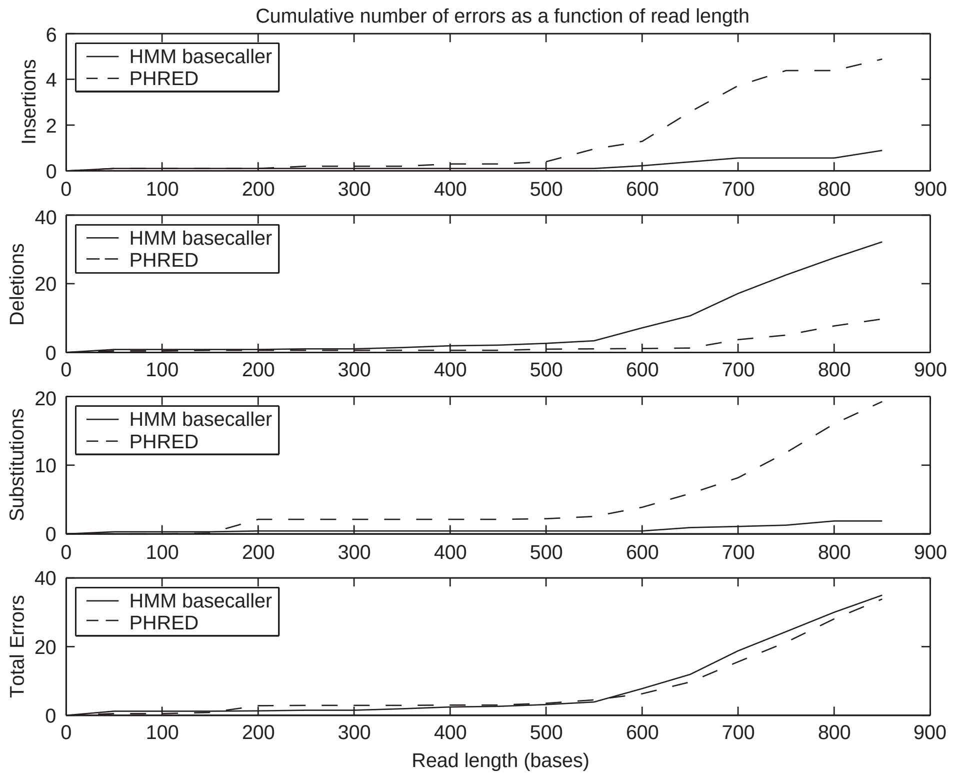


Fig. 7. Comparison of PHRED to the HMM basecaller. The total error performance is comparable.

6. Future work

Improving basecalling results is very important in the sequencing community as it saves significantly on sequencing cost. The results are promising, but further research is needed to improve performance even further and make HMMs useful in the community. In this section we try to identify areas where further research might be fruitful.

6.1. Confidence measures

Although HMMs provide a direct probabilistic interpretation of the results in terms of likelihoods, this interpretation is different than the PHRED confidence scores described in [4]. Since the biological community is accustomed to these scores being reported, acceptance of an alternative basecaller depends on providing confidence measures on a similar scale. For example, sequence assembly methods such as [1,2] rely on these scores to evaluate consensus sequences. Thus, further work is needed to provide intuition and tools to compare these two measures.

Such work is also necessary for other basecalling methods. A common language to report error estimates has several potential benefits. For example, it will be possible to combine results from different basecallers in order to assemble a sequence. It is also possible that one error estimate for each base call is not enough, since there are several types of errors. Certain basecallers might be biased towards performing one type of error, as was evident in our results. Making error estimates for each kind of error allows a basecaller to report such a bias to the assembly algorithm.

6.2. Model topology

Our system produced a significant number of deletion errors using the simple model to perform recognition. Although we believe that the alternative models we propose will improve the results, we still need to test them. Even with these models, we have not exhausted the model space. Other models might exhibit even better performance and further research is needed to determine them.

6.3. Features selection and emissions model

In this work we did not try to optimize our feature selection process. Instead, we relied on the neural network to estimate the emission probability based on a window of the data. With appropriate features and emissions model the data might be better represented, improving performance or reducing complexity. For example, we might be able to use Gaussian mixture models for emission densities and thus achieve faster convergence in training, or Support Vector Machines to better describe the data.

6.4. Preprocessing

Another step of the process we have not examined is the preprocessing of the electropherogram before it is fed to the basecaller. Several aspects of the preprocessing steps could be better tuned for use with an HMM basecaller. For example, the preprocessor low-pass filters the data to remove the noise and make electropherograms look better to a human reader. Certain algorithms might rely on a noise-free electropherogram to operate, but this process might be suboptimal for other algorithms. In fact, low-pass filtering has the potential of destroying subtle but important transition features in the data by smoothing them out. Our basecaller does not depend on a noise-free signal to operate since a noise model can be incorporated in the state emission statistics. Thus, it might be desirable to modify the preprocessing steps to improve performance.

6.5. Extensions

The potential applications of HMMs in other areas of biology, such as SNP detection and DNA fingerprinting are very promising. Furthermore, similar models can be used for protein sequencing, aiding the field of proteomics. We believe that the potential applications in other fields are numerous.

7. Conclusions

We believe we have demonstrated the suitability and the potential of hidden Markov models as a basecalling tool. We presented a very simple model that matches the overall performance of PHRED, at least in a preliminary evaluation. Still, our model provides significant room for improvement, especially in terms of deletion errors. Thus, we proposed alternative models that should improve performance. Nevertheless, a careful validation of the results is necessary, using a variety of data, and a larger sample. Finally, we identified promising areas of research that might be fruitful in improving the performance of the algorithm, as well as extensions to other areas.

Acknowledgements

The authors would like to acknowledge the support of the following grants: National Institutes of Health Grant HG01389, Air Force Office of Scientific Research Grant F49620-98-1-0235, and the National Institute of Justice award 98-LB-VX-K022.

References

[1] S. Batzoglou, et al., ARACHNE: a whole-genome shotgun assembler, Genome Res. 12 (1) (2002) 177–189.
[2] P. Green, PHRAP, http://www.phrap.org.
[3] B. Ewing, et al., Base-calling of automated sequencer traces using *Phred*. I, Accuracy assessment, Genome Res. 8 (3) (1998) 175–185.
[4] B. Ewing, P. Green, Base-calling of automated sequencer traces using *Phred*. II. Error probabilities, Genome Res. 8 (3) (1998) 186–194.
[5] D. Nelson, Improving DNA sequencing accuracy and throughput, in: Genetic Mapping and DNA Sequencing, Springer, New York, 1996, pp. 183–206.
[6] N.M. Haan, S.J. Godsill, Modelling electropherogram data for DNA sequencing using variable dimension MCMC, in: Acoustics, Speech, and Signal Processing, 2000, ICASSP '00, Proceedings, 2000 IEEE International Conference, Vol. 6, Instanbul, Turkey, 2000, pp. 3542–3545.
[7] M. Pereira, et al., Statistical learning formulation of the DNA base-calling problem and its solution using a Bayesian EM framework, Discrete Appl. Math. 104 (1–3) (2000) 229–258.
[8] P. Boufounos, et al., Hidden Markov models for DNA sequencing, in: Proceedings of Workshop on Genomic Signal Processing and Statistics (GENSIPS 2002), Raleigh, NC, USA, 2002.
[9] L.R. Rabiner, A tutorial on hidden Markov models and selected applications in speech recognition, Proc. IEEE 77 (2) (1989) 257–286.
[10] P. Boufounos, Signal processing for DNA sequencing, M.Eng. Thesis, MIT press, Cambridge, MA, June 2002.
[11] J. Hennebert, et al., Estimation of global posteriors and forward–backward training of hybrid HMM/ANN systems, in: Eurospeech'97, Rhodes, Greece, 1997, pp. 1951–1954.
[12] T.P. Minka, Expectation-maximization as lower bound maximization, http://web.media.mit.edu/tpminka/papers/em.html, revised November 99 [Accessed October 2003] (November 98).
[13] J.M. Bowling, et al., Neighboring nucleotide interactions during DNA sequencing gel electrophoresis, Nucleic Acids Res. 19 (11) (1991) 3089–3097.
[14] J.M. Short, et al., Lambda ZAP: a bacteriophage lambda expression vector with in vivo excision properties, Nucleic Acids Res. 16 (15) (1998) 7583–7600.