Chapter 4

A method for estimating optimal parameters for a GAZE model

4.1 Introduction

A key property of the GAZE system is the ability to include signal and content information of arbitrary types from multiple sources. This was evident in the GAZE_EST model of the last chapter, where separate segment types corresponding respectively to regions of high coding potential and EST database hits both contributed to the score for candidate coding exons. For such an approach to work effectively, the evidence of each type must be *weighted* appropriately. For the GAZE_EST model, this problem was addressed in a rather *ad hoc* manner, by applying a scaling factor to the scores of the "EST_match" segments to make them approximately of the same order as the segments of high coding potential. This scheme worked well in this case, but in general it will be difficult to to obtain optimal evidence weightings in this way. Far more desirable would be a way to automatically derive optimal weights for the segments used in a model.

This chapter describes my work on an automated method for obtaining optimal weights for the scores of the various pieces of evidence referred to in a GAZE model. The idea is to obtain the set of weights for which the prediction accuracy of the model is as high as possible.

I start by modifying the GAZE scoring function to accommodate weights for each type of evidence. I then go on to describe the design and implementation of a method for obtaining the optimal set of weights for a GAZE model with respect to a set of training sequences with known gene structures. The technique is inspired by the work of Stormo and Haussler [109] and Krogh [68] (amongst others) and therefore contrast it with other kinds of parameter estimation in the field of gene prediction. The chapter ends with an example application, namely training the evidence weights for the GAZE_EST gene prediction system for *C. elegans* sequences (see chapter 3). A more detailed examination of the performance of the method is presented in the following chapter, whereby it is applied to the implementation of a GAZE gene prediction system for vertebrate sequences.

4.2 Evidence weighting in GAZE

4.2.1 Optimally parsing a sequence according to weighted evidence

In 1994, Stormo and Haussler published a method for *parsing* a sequence into regions of "intron" and "exon" using multiple types of weighted evidence for each of introns and exons [109]. In their method, exons and introns are scored according to the following scheme:

$$T_E(i,j) = \sum_{\mu \in M} w_\mu C_\mu(i,j)$$

$$T_I(i,j) = \sum_{\kappa \in K} w_\kappa C_\kappa(i,j)$$

M is a collection of types of *evidence* for exons; $C_\mu(i,j)$ is the raw "score" for evidence-type μ over the region $i \ldots j$ of the sequence; w_μ is a real number *weight*

for evidence-type μ. K, $T_\kappa(i,j)$ and w_κ are similarly defined for introns. The score of a *parse* of a query sequence into exons and introns is simply obtained by addition of the appropriate values from the T matrices. For example, for a sequence 1000 base pairs long, the parse (1-50,intron):(51-75,exon):(76-600, intron):(601-950, exon):(951-1000, intron) would be scored as:

$$T_I(1,50) + T_E(51,75) + T_I(76,600) + T_E(601,950) + T_I(951,1000)$$

The authors show how the highest scoring parse can be obtained by dynamic programming. They then go onto define a probability distribution over all possible parses, and present a procedure for obtaining the set of weights that maximises the probabilities of the correct parses of a set of training sequences. The method described in the following sections has at its core a generalisation of this procedure to make it applicable to GAZE, where the model of gene structure is not known in advance. I start though by describing how evidence weighting is accommodated in the GAZE scoring function.

4.2.2 Accommodating weights in the GAZE scoring function

A weight is attached to every piece of evidence that contributes to the score; that is, each feature, segment and length penalty function. Each weight is a floating point number by which the score of the associated element is scaled, and features (and segments) of the same type are scaled by the same weight. Intuitively therefore, the weights can be viewed as a way of ascribing a relative importance firstly to each element of the model (e.g. splice site versus translation start sites) and secondly to different types of evidence supporting the same region type (e.g. coding segments versus database matches for potential protein coding exons). I refer to the complete set of weights for a given model as $\mathbf{w}$.

In order to accommodate the weights, I start by reformulating the GAZE scoring (section 2.4). Given again an ordered list $\phi = \phi_1, \phi_2, \ldots, \phi_n$ of features defining a valid gene structure according to a GAZE model (with ϕ_0 and ϕ_{n+1} denoting

respectively the special features "BEGIN" and "END" marking the beginning and the end of the sequence; see chapter 2), then the weighted score of ϕ, $E(\phi, \mathbf{w})$ is calculated as:

$$E(\phi, \mathbf{w}) = \sum_{i=0}^{n} T(\phi_i, \phi_{i+1}, \mathbf{w}) \tag{4.1}$$

$$\begin{aligned} T(\phi_i, \phi_j, \mathbf{w}) &= Seg_{t(\phi_i) \to t(\phi_j)}(l(\phi_i), l(\phi_j), \mathbf{w}) \\ &+ Len_{t(\phi_i) \to t(\phi_j)}(l(\phi_j) - l(\phi_i) + 1, \mathbf{w}) \\ &+ Loc(\phi_j, \mathbf{w}) \end{aligned} \tag{4.2}$$

The function $T(\phi_i, \phi_j, \mathbf{w})$ is calculated as a composite score for the region $\phi_i \to \phi_j$, comprising components for segment, length penalty and local feature scores as before (see equations 2.1, 2.2). The difference is that weights for the various types of evidence are included in T.

I construct a mapping W from the types of the features, segments and length penalty functions to the elements of $\mathbf{w}$ such that each of the former is associated with exactly one of the latter. For example, $W(\mathbf{w}, t(\phi_i))$ is the element of $\mathbf{w}$ that is the weight for feature ϕ_i and all other features of the same type. Given this mapping, the feature, segment and length-penalty components of the score are scaled in the following way:

Feature score weighting

The weighted local score for a feature ϕ_i is calculated as the given score scaled by the weight :

$$Loc(\phi_i, \mathbf{w}) = g(\phi_i) W(\mathbf{w}, t(\phi_i)) \tag{4.3}$$

Segment score weighting

Recall from equation 2.3 that the segment score for a region is a sum of scores for each segment qualifier given in the model rule for the region. If again ψ^q is the relevant subset of segments for segment qualifier q, and $t(q)$ is the type of those segments[1], then the weighted segment score is calculated as:

$$Seg_{src \to tgt}(x, y, \mathbf{w}) = \sum_{q \in Q_{src \to tgt}} Seg^Q(\psi^q, x, y) W(\mathbf{w}, t(q)) \tag{4.4}$$

The additive nature of both specific segment scoring strategies (equations 2.4 and 2.5) means that this weighting can be applied in practice directly to the given scores for each segment in advance (according to type), as was the case for the features above.

Length penalty weighting

Finally, the weight for a length penalty function is a simple scaling factor applied to each penalty value:

$$Len_{src \to tgt}(x, \mathbf{w}) = Len_{src \to tgt}(x) W(\mathbf{w}, t(Len_{src \to tgt})) \tag{4.5}$$

where $t(Len_{src \to tgt})$ is an identifier for the length penalty function used in the rule $src \to tgt$. This approach assumes that the *shape* of the function has already been determined by some sort of inversion of a frequency-of-occurrence histogram. An alternative would be to model each function as a series of constants, each subject to a separate scaling factor. This would potentially provide a powerful method for the simultaneous estimation of the shape and weight of each penalty function. Doing this however would greatly increase the number of free variables of the system, and I will not consider it further here.

[1] all relevant segments for a Segment Qualifier must have the same type due to the compulsory type constraint

In practice, specific values for the weights are defined in the configuration file, via a "mul" attribute attached to the the declaration of each element of the gene structure model. When GAZE is used in prediction mode, the given values are used to scale the scores of the appropriate model element in the way described above before the dynamic programming is performed. When used in parameter-estimation mode, the given values define the starting value of the function to be optimised.

4.3 Two approaches to obtaining an optimal set of weights

The aim of the method is to obtain the set weights $\mathbf{w}^{opt}$ that maximises the gene prediction accuracy in a set of training sequences for which the gene structures are known. The natural approach would be to design a function of the weights that represents the gene prediction accuracy in the training sequences, and then maximise this function with respect to the weights. A problem with this method is that the standard measures of accuracy are not continuous functions of the weights. This is because "accuracy", as it has been defined so far, depends only upon the highest scoring gene structure, and a small change in the weights can lead to a large change in the optimal gene structure. By using the posterior probabilities of chapter 2 however, it is possible to construct an accuracy function that is continuous in the weights. I have implemented two accuracy functions based on these posterior probabilities. The degree to which they give rise to an effective set of weights is investigated in the following chapter. The remainder of this chapter focuses on the design, implementation and optimisation of the functions themselves.

4.3.1 Maximum Likelihood

This method involves identifying the set of parameters that maximises the log-probability of the correct (i.e. annotated/confirmed) gene structures in a set of training sequences. Stormo and Haussler [109] showed how to do this in the context of a gene prediction system based on weighted evidence, and their method is directly

applicable to the GAZE framework. Recall equation 2.9 which defines a probability distribution over all possible valid gene structures (given a model). If there are K training sequences, and the *correct* gene structure in training sequence k is ϕ^{k*}, then the ML approach is to maximise the following as a function of $\mathbf{w}$:

$$\alpha^{\mathrm{ML}}(\mathbf{w}) = \sum_{k=1}^{K} \ln \mathrm{P}(\phi^{k*}|\mathbf{w}) \tag{4.6}$$

The posterior probability of the correct gene structure is thus made to be as high as possible.

4.3.2 Maximal Feature Discrimination

By maximising the posterior probability of the correct gene structure, we at the same time minimise the summed probabilities of all *incorrect* structures. Intuitively, this appears desirable, but the drawback is that all of the incorrect gene structures are considered equally "incorrect". This is clearly not a good representation of gene prediction accuracy; some candidate structures, although not completely correct, will be closer to the correct structure than others. By maximising the probability of the correct structure only, we may (and often do) increase the probability of individual incorrect structures so that they score better than the correct structure, and the highest scoring structure is even "less correct" than before.

The other main problem of the ML approach as presented above is that it relies heavily on the knowledge of a single complete correct gene structure. We know that many sequences have more than one structure that can be described as correct, due to alternative splicing. It will also generally not be the case that we have accurate data about all the feature types present in a model. In the previous chapter, I showed how a generic model of gene structure could be extended to incorporate *trans*-splice site candidates (via acceptor splice site predictions) and transcription initiation and termination candidates (via EST information). The predictions made by the extended model include regions defined by the additional features, such as untranslated regions (where evidence was present). The assessment of the accuracy

of these predictions was not completely straightforward, because the "correct" gene structures with which I was comparing included only the protein-coding part of each gene. This is true of the majority of benchmark sets used to train and assess the accuracy of gene prediction programs. I therefore took the approach of considering only the protein-coding part of each predicted structure in the assessment, and this worked adequately. Such a method will not work for the maximum-likelihood training function above however; supplying a "correct" gene structure consisting of a proper subset of the feature-types referred to in a GAZE model will cause the weights for the other feature types to tend to negative infinity. We therefore need a more general way of dealing with correct gene structures for which only the regions involving specific feature types are known.

To address both of these problems, I have implemented a new objective function of $\mathbf{w}$ that maximises the posterior probabilities of the *features* that are part of the correct structure, while minimising the same for those that are not. In addition, I provide a way of ignoring features of particular types in the optimisation. For consistency with above, I work in log probabilities. If $\phi_1^k \ldots \phi_{n(k)}^k$ this time is the list of candidate features for training sequence k (where the special features ϕ_0^k and $\phi_{n(k)+1}^k$ mark the beginning and end of the sequence as usual), then the function is:

$$\alpha^{\mathrm{MFD}}(\mathbf{w}) = \sum_{k=1}^{K} \sum_{i=0}^{n(k)+1} r(\phi_i^k)[c(\phi_i^k) \ln \mathrm{P}(\phi_i^k|\mathbf{w}) + (1 - c(\phi_i^k)) \ln(1 - \mathrm{P}(\phi_i^k|\mathbf{w}))] \quad (4.7)$$

$$\begin{aligned} c(\phi_i^k) &= 1 \quad \text{if} \quad \phi_i^k \in \phi^{k*} \\ &= 0 \quad \text{otherwise} \end{aligned} \quad (4.8)$$

$$\begin{aligned} r(\phi_i^k) &= 1 \quad \text{if} \quad t(\phi_i^k) \notin Ignore \\ &= 0 \quad \text{otherwise} \end{aligned} \quad (4.9)$$

The ideal result of maximising this function is a set of weights for which *all* gene structures use as many correct features and as few incorrect features as possible. We might expect, by optimising this function, to perform better on feature-based measures of gene prediction accuracy. I refer to this method as Maximal Feature Discrimination (MFD).

The problem of knowing only portions of the correct gene structure that involve specific feature types is also addressed by this function: in its computation, we simply ignore the features that are not of one of these specific types. We therefore maximise the probabilities of the features which we know to be correct and minimise the probabilities of those that are known to be incorrect; the probabilities of the other, ignored features are unconstrained. It is important to note that this is *not* a general strategy for making use of partial correct gene structures; if a feature type is considered relevant for the computation of the above function, then all of the features of that type present in the given "correct" gene structure are considered correct, and, importantly, all other candidate features of that type are considered incorrect. Hence the method is for dealing with feature *types* that are absent from the the correct gene structures, not individual features.

4.4 Optimising the objective functions by gradient descent

There are many published methods for maximising a multi-dimensional function such as those described above (see [88] for a detailed review). I have chosen to use a gradient-based method. Each evaluation of the functions above requires dynamic programming (see chapter 2), and a gradient-based method should give rise to fewer function evaluations than a method that does not employ such information, particularly when the number of dimensions is high.

I first describe the maximisation scheme that was used in terms of an anonymous multi-dimensional objective function of a vector of parameters, $\alpha(\mathbf{w})$. I go on to show

in the next section how the derivatives of the two objective functions described in the previous section can be obtained, thus making the method directly applicable to GAZE.

4.4.1 A conjugate gradient descent method

To be consistent with other literature on function optimisation, I present this as a minimisation algorithm. Maximisation of a function is equivalent to minimisation of the negative of the function.

We wish to find the values for the elements of $\mathbf{w}$ for which the function $\alpha(\mathbf{w})$ is at its minimum. The method of *steepest descent* begins with an arbitrary point $\mathbf{w}^0$, and then repeatedly (a) computes a vector of partial derivatives $\nabla\mathbf{w}^t$ at the current point $\mathbf{w}^t$ and then (b) minimises along the line $\mathbf{w}^t + \eta\nabla\mathbf{w}^t$ to obtain a new current point $\mathbf{w}^{t+1}$:

$$\mathbf{w}^{t+1} = \min_{\eta} \alpha(\mathbf{w}^t + \eta\nabla\mathbf{w}^t)$$

The procedure is terminated when some stopping condition is satisfied, for example when the change in function value is small enough.

The "line minimisation" step is the most complicated part of the procedure and involves two basic steps: (a) identifying a range for η within which the minimum must lie (i.e. *bracketing* the minimum) and (b) successively narrowing this region until it is small enough to define the minimum to within a tolerable error based on the precision of the machine (i.e. a *section* search). For both of these steps, I have implemented multi-dimensional versions of the algorithms described in the book by Press *et. al.* [88]. In particular, the section search method was originally described by Brent [16].

As explained in [88], the problem with the simple steepest descent method is that for many common functions, it will perform many small steps in descending a long narrow valley-like function. This is because consecutive line minimisations are necessarily in orthogonal directions, so the process zig-zags from side to side.

I have therefore implemented a variation of the steepest descent method proposed by Fletcher and Reeves [43] and later modified by Polak and Ribiere [87]. The idea of their method is to minimise not in the direction of the gradient at the new point, but in a direction that does not interfere with any of the previous directions travelled. The concept of non-interfering directions is formalised by Press *et. al.* [88] as *conjugacy*. Two vectors $\mathbf{u}$ and $\mathbf{v}$ are said to be *conjugate* (i.e. non-interfering) if the following condition holds:

$$\mathbf{u}.\mathbf{A}.\mathbf{v} = 0$$

where $\mathbf{A}$ is the matrix of second partial derivatives. The authors show that by scaling and subtracting the last direction travelled $\mathbf{u}$ from the gradient at the new point, a new direction conjugate to $\mathbf{u}$, $\mathbf{v}$ can be obtained without explicit calculation of this matrix. The resulting method belongs to a sub-family of similar methods known in the literature as *conjugate gradient descent*.

The gradient descent method as I have described it has the desirable property that the process is divided into "chunks" (line minimisations), between which progress can be monitored and assessed if necessary.

4.5 Calculating the gradient by dynamic programming

To use conjugate gradient descent, or any gradient-based method, it is necessary to be able to obtain, for any function point, the partial first-derivatives with respect to each variable. Furthermore, it is necessary to do this efficiently, as the reason for using gradient information in the first place is to keep the time taken to reach the minimum as small as possible. In this section, I show how the derivatives of both of the functions proposed in section 4.3 can be efficiently computed.

4.5.1 The derivative of the ML function

The partial derivative of the Maximum Likelihood function (equation 4.6) with respect to single variable w in $\mathbf{w}$ is obtained by expressing the log probability of the

correct gene structure in terms of its constituent components. Since the function comprises a sum over the training sequences, I simplify the notation by assuming a single training sequence with correct gene structure denoted by ϕ^*.

$$\begin{aligned}\frac{\partial\,\alpha^{\mathrm{ML}}(\mathbf{w})}{\partial\,w} &= \frac{\partial\,\ln P(\phi^*|\mathbf{w})}{\partial\,w} \\ &= \frac{\partial}{\partial\,w}\ln\frac{\mathrm{e}^{E(\phi^*)}}{Z} \\ &= \frac{\partial\,E(\phi^*)}{\partial\,w} - \frac{Z'}{Z}\end{aligned} \tag{4.10}$$

where Z is the partition function (given by equation 2.8), and Z' is its derivative. The derivative of the score of the correct gene structure can be computed simply as the sum of the derivatives of the components. Given again a list $\phi_0, \phi_1, \ldots, \phi_{n+1}$ defining the correct gene structure, we have:

$$\frac{\partial\,E(\phi,\mathbf{w})}{\partial\,w} = \sum_{i=0}^{n}\frac{\partial}{\partial\,w}T(\phi_i,\phi_{i+1},\mathbf{w}) \tag{4.11}$$

$$\begin{aligned}\frac{\partial}{\partial\,w}T(\phi_i,\phi_j,\mathbf{w}) &= \frac{\partial}{\partial\,w}Seg_{t(\phi_i)\to t(\phi_j)}(l(\phi_i),l(\phi_j),\mathbf{w}) \\ &+ \frac{\partial}{\partial\,w}Len_{t(\phi_i)\to t(\phi_j)}(l(\phi_j)-l(\phi_i)+1,\mathbf{w}) \\ &+ \frac{\partial}{\partial\,w}Loc(\phi_j,\mathbf{w})\end{aligned} \tag{4.12}$$

The feature, segment and length-penalty components are each themselves linear sums of terms involving at most one variable, so it is straightforward to obtain their derivatives. In particular, the derivative with respect to parameter w will be zero for all components of the score that are not relevant for that weight, and equal to the sum of raw (unweighted) scores for the component otherwise.

If Φ is the space of all gene structures, the derivative of the partition function can be expressed as follows:

$$Z' = \sum_{\phi\in\Phi}\frac{\partial\,E(\phi)}{\partial\,w}\mathrm{e}^{E(\phi)} \tag{4.13}$$

It can thus be thought of as an "weighted average" of the derivatives of all gene structures. A different but related term occurs in the gradient of the Maximal Feature Discrimination objective function. I will show how these quantities can be calculated in section 4.5.3, after deriving the gradient of the MFD function.

4.5.2 The derivative of the MFD function

For the Maximal Feature Discrimination function (given by equation 4.7) I again simplify notation by assuming a single training sequence with a complete list of candidate features $\phi_0 \ldots \phi_{n+1}$.

$$\begin{aligned}
\frac{\partial\, \alpha^{\mathrm{MFD}}(\mathbf{w})}{\partial\, w} &= \frac{\partial}{\partial\, w} \sum_{i=0}^{n+1} r(\phi_i)[c(\phi_i) \ln P(\phi_i|\mathbf{w}) + (1 - c(\phi_i)) \ln(1 - P(\phi_i|\mathbf{w}))] \\
&= \sum_{i=0}^{n+1} r(\phi_i) \left[\frac{c(\phi_i)}{P(\phi_i|\mathbf{w})} \frac{\partial\, P(\phi_i|\mathbf{w})}{\partial\, w} - \frac{1 - c(\phi_i)}{1 - P(\phi_i|\mathbf{w})} \frac{\partial\, P(\phi_i|\mathbf{w})}{\partial\, w} \right] \\
&= \sum_{i=0}^{n+1} r(\phi_i) \left[\frac{c(\phi_i) - P(\phi_i|\mathbf{w})}{P(\phi_i|\mathbf{w})(1 - P(\phi_i|\mathbf{w}))} \frac{\partial\, P(\phi_i|\mathbf{w})}{\partial\, w} \right] \qquad (4.14)
\end{aligned}$$

In order to obtain the derivatives of the posterior feature probabilities, I first define Z_i to be the sum of exponentiated scores of all gene structures that include feature ϕ_i:

$$Z_i = \sum_{\phi \in \Phi : \phi_i \in \phi} \mathrm{e}^{E(\phi)} \qquad (4.15)$$

The term Z_i can be thought of as an "i-restricted" partition function, and dividing it by the unrestricted partition function gives the posterior probability for feature ϕ_i (see equation 2.15). The derivative of the posterior probability of ϕ_i can now be derived:

$$\begin{aligned}
\frac{\partial\, P(\phi_i|\mathbf{w})}{\partial\, w} &= \frac{\partial}{\partial\, w} \frac{Z_i}{Z} \\
&= Z^{-1} \frac{\partial\, Z_i}{\partial\, w} - Z_i Z^{-2} \frac{\partial\, Z}{\partial\, w} \\
&= P(\phi_i|\mathbf{w}) \left(\frac{Z_i'}{Z_i} - \frac{Z'}{Z} \right) \qquad (4.16)
\end{aligned}$$

The derivative of the i-restricted partition function, Z'_i can be expressed in terms of a "weighted average of derivatives" in the same way as the unrestricted function:

$$Z'_i = \sum_{\phi \in \Phi : \phi_i \in \phi} \frac{\partial E(\phi)}{\partial w} \mathrm{e}^{E(\phi)} \tag{4.17}$$

Expressing the derivatives of the i-restricted and unrestricted partition functions in this way is the first step towards the design of a single method for the computation of both, shown next.

4.5.3 Computing the weighted average of the derivatives

The aforementioned Stormo and Haussler article [109] includes a sketch of a dynamic programming procedure for the efficient computation of a quantity analogous to Z' above, i.e. the gradient of the partition function. A simple extension of their method allows for the simultaneous computation of the gradients of both the i-restricted and unrestricted partition functions.

The method relies firstly upon the fact that a gene structure $\phi_1 \dots \phi_n$ can be decomposed into two partial gene structures $\phi_1 \dots \phi_i$ and $\phi_i \dots \phi_n$, with the score of the complete structure being the sum of the two partial structures[2]. If we denote the set of all partial gene structures *ending* with feature ϕ_i as $\Phi^{\cdot\cdot\phi_i}$, and likewise the set of all partial structure *beginning* with feature ϕ_i as $\Phi^{\phi_i\cdot\cdot}$, then Z'_i can be expressed in terms of all partial structures beginning and ending (respectively) with feature ϕ_i:

$$\begin{aligned} Z'_i &= \frac{\partial}{\partial w} \sum_{\phi^x \in \Phi^{\cdot\cdot\phi_i}} \sum_{\phi^y \in \Phi^{\phi_i\cdot\cdot}} \mathrm{e}^{E(\phi_x)} \mathrm{e}^{E(\phi_y)} \\ &= \sum_{\phi \in \Phi^{\cdot\cdot\phi_i}} \mathrm{e}^{E(\phi)} \sum_{\phi \in \Phi^{\phi_i\cdot\cdot}} \frac{\partial E(\phi)}{\partial w} \mathrm{e}^{E(\phi)} \end{aligned}$$

[2] the asymetry of the scoring function is helpful here; the local score for feature ϕ_i is included in the partial structure for which it forms the end, but not for the structure for which it forms the start

$$
\begin{aligned}
&+ \sum_{\phi \in \Phi^{\phi_i \cdot \cdot}} \mathrm{e}^{E(\phi)} \sum_{\phi \in \Phi^{\cdot \cdot \phi_i}} \frac{\partial\, E(\phi)}{\partial\, w} \mathrm{e}^{E(\phi)} \\
= \; & f(i) \sum_{\phi \in \Phi^{\phi_i \cdot \cdot}} \frac{\partial\, E(\phi)}{\partial\, w} \mathrm{e}^{E(\phi)} + b(i) \sum_{\phi \in \Phi^{\cdot \cdot \phi_i}} \frac{\partial\, E(\phi)}{\partial\, w} \mathrm{e}^{E(\phi)} \qquad (4.18)
\end{aligned}
$$

The final step follows from chapter 2, where the exponentiated scores of partial structures ending and beginning with feature ϕ_i were computed respectively as $f(i)$ and $b(i)$ (equations 2.11, 2.14).

All that remains is to obtain the weighted average derivatives for structures beginning and ending (respectively) at ϕ_i. Let $u^f(i, w)$ be the weighted average derivative of all partial structures ending with feature ϕ_i, with respect to parameter w. Each of these partial structures can be decomposed further into a partial structure ending at ϕ_j $(j < i)$, and the final region $\phi_j \rightarrow \phi_i$. The score of this region can be calculated independently as $T(\phi_j, \phi_i, \mathbf{w})$, which allows us to derive a dynamic programming recurrence for u^f:

$$
\begin{aligned}
u^f(i, w) &= \sum_{\phi \in \Phi^{\cdot \cdot \phi_i}} \frac{\partial\, E(\phi)}{\partial\, w} \mathrm{e}^{E(\phi)} \\
&= \sum_{0 \leq j < i} \sum_{\phi \in \Phi^{\cdot \cdot \phi_j}} \left(\frac{\partial\, E(\phi)}{\partial\, w} + \frac{\partial\, T(\phi_j, \phi_i, \mathbf{w})}{\partial\, w} \right) \mathrm{e}^{E(\phi)} \mathrm{e}^{T(\phi_j, \phi_i, \mathbf{w})} \\
&= \sum_{0 \leq j < i} \mathrm{e}^{T(\phi_j, \phi_i, \mathbf{w})} \left(\sum_{\phi \in \Phi^{\cdot \cdot \phi_j}} \frac{\partial\, E(\phi)}{\partial\, w} \mathrm{e}^{E(\phi)} + \frac{\partial\, T(\phi_j, \phi_i, \mathbf{w})}{\partial\, w} \sum_{\phi \in \Phi^{\cdot \cdot \phi_j}} \mathrm{e}^{E(\phi)} \right) \\
&= \sum_{j < i} \mathrm{e}^{T(\phi_j, \phi_i, \mathbf{w})} \left(u^f(j, w) + \frac{\partial\, T(\phi_j, \phi_i, \mathbf{w})}{\partial\, w} f(j) \right) \qquad (4.19)
\end{aligned}
$$

The vectors for each parameter u^f can thus be populated by dynamic programming (with $u^f(0, w) = 0$), and the weighted average derivative over all structures, Z', is obtained directly as $u^f(n + 1, w)$. It is straightforward to construct an analogous "backwards" matrix where $u^b(i, w)$ denotes the weighted average of the derivatives of partial structures *beginning* with feature i :

$$
u^b(i, w) = \sum_{i < k \leq n+1} \mathrm{e}^{T(\phi_i, \phi_k, \mathbf{w})} \left(u^b(k, w) + \frac{\partial\, T(\phi_i, \phi_k, \mathbf{w})}{\partial\, w} b(k) \right) \qquad (4.20)
$$

The derivative of the i-restricted partition function then reduces to the following:

$$Z'_i = f(i)u^b(i,w) + b(i)u^f(i,w) \tag{4.21}$$

Drawing all of these elements together, the derivatives of both the Maximum Likelihood and Maximal Feature Discrimination functions can now be written down in terms of elements that are directly computable by the methods presented here and in chapter 2:

$$\frac{\partial\,\alpha^{\mathrm{ML}}(\mathbf{w})}{\partial\,w} = \left(\sum_{\phi_x \to \phi_y \in \phi^*} \frac{\partial\,T(\phi_x,\phi_y,\mathbf{w})}{\partial\,w}\right) - \frac{u^b(0,w)}{b(0)} \tag{4.22}$$

$$\frac{\partial\,\alpha^{\mathrm{MFD}}(\mathbf{w})}{\partial\,w} = \sum_{i=0}^{n+1} r(\phi_i)\left(\frac{c(\phi_i) - P(\phi_i|\mathbf{w})}{1 - P(\phi_i|\mathbf{w})}\right)\left(\frac{u^b(i,w)}{b(i)} + \frac{u^f(i,w)}{f(i)} - \frac{u^b(0,w)}{b(0)}\right) \tag{4.23}$$

4.6 Implementation issues

4.6.1 Numerical stability

As in chapter 2, the calculation of the derivatives of the i-restricted and unrestricted partition functions is performed in log-space to cope with limitations in machine precision:

$$U(0) = -\infty$$

$$U^f(i,w) = \ln \sum_{0 \le j < i} \left(\mathrm{e}^{T(\phi_j,\phi_i,\mathbf{w}) + U^f(j,w)} + \frac{\partial\,T(\phi_j,\phi_i,\mathbf{w})}{\partial\,w}\mathrm{e}^{F(j)}\right)$$

Unlike the log-space computation of the forward and backward variables presented in chapter 2, the sum above is not guaranteed to be positive. This is due to the unexponentiated term in the sum, the derivative of $T(\phi_i,\phi_k,\mathbf{w})$, which is often negative. Additional book-keeping is therefore necessary in which I take the log of *absolute* value of the sum, keeping track of those (i,w) for which $\mathrm{e}^{U^f(i,w)}$ is in reality

negative and reversing the sign of the exponentiation of the $U^f(j, w)$ in the equation above for those cases.

The Maximal Feature Discrimination function and its derivative are problematic to implement in practice due to the fact they are both undefined in the cases of a correct feature having posterior probability of 0 or an incorrect feature having a posterior probability of 1. Such situations are not impossible, and can be approached arbitrarily closely if the feature set is incompatible with the correct gene structure. I therefore took the natural approach of replacing $\log 0$ and $\frac{-x}{0}$ with a large negative numbers. Both the value and the gradient of the function are somewhat inaccurate therefore at extreme values for the weights, but since such extremities are typically only encountered whilst bracketing the minimum, this is unimportant.

It is often the case that functions with complex behaviour such as those being optimised here have many local minima; the local minimum obtained depends greatly on the starting conditions of the procedure, in this case the initial values for the weights. There are two techniques commonly used to address this problem. The first is *simulated annealing* [63], whereby a gradually diminishing level of "noise" is added to the parameter values to give the function a stochastic opportunity to escape from a local minimum. I have implemented the other common technique which is to perform the optimisation several times from different starting points, and choose the set of parameters that give the overall lowest function value at termination. This undirected strategy works well in my case, because there is usually some approximate idea of what the weights should be for a given model. By starting the process from such weights we increase the chance of reaching a global minimum, or at least a set of weights that work well. Re-optimising with different starting conditions and obtaining a solution that is the same or larger is verification that the minimum obtained is likely to be global.

4.6.2 Parameter tying

In many cases, it makes sense that the score for two (or more) different pieces of evidence should have the same weight. This is true for example when different feature or segment types are used for the forward and reverse strand of a sequence. I have therefore included a mechanism for indicating that the weights for two or more model elements should be *tied* together during the optimisation process. Parameter tying is a common technique in the application of Hidden Markov Models to speech recognition. It also has widespread use in the field of neural networks [11], where is known as *weight sharing*.

In the GAZE framework, tying is achieved simply via the function W that maps feature, segment and length penalty types onto weights. For the description earlier, I implied that this mapping was one-to-one. By making it many-to-one, more than one evidence type can map to the same weight. In practice, this means that a weight tied to many pieces of evidence is updated according to a sum of contributions from each.

It is also worth saying here that although the framework presented assumes for flexibility that the scores for *every* type of evidence referred to in a GAZE model is associated with a weight (with some possibly associated with the same weight, as explained above), this need not be the case. The optimisation system has a mechanism to allow the user to specify which types of evidence are to be treated as having variable weight. This has no effect on the equations presented above; the derivative of all functions with respect to non-variable weights is zero.

4.6.3 Time and memory usage

Most of the issues discussed in chapter 2 concerning the time and memory usage of GAZE are not different here and are addressed in the same way; the algorithm for the gradient calculations is ostensibly the same as the forward and backward algorithms, and in fact can be implemented as extensions of both. The run-time is increased by a constant factor of up to 10 however, so it is therefore sensible to only

calculate the gradient when necessary.

The optimisation algorithm can be summarised as a series of gradient calculations, each followed by a minimisation along the line defined by the current point and the new direction (a function of the new gradient and previous gradients). The first of these steps involves a single forward-backward-with-gradient run over the training sequences. The second, line minimisation requires a succession of function evaluations. The section search algorithm by Brent [16] has been extended by Press [88] to make use of derivative information with the aim of reducing the number of evaluations necessary in the line minimisation. However, I found the overhead in calculating a forward-backward-with-gradient for *every* function evaluation as opposed to a straightforward forward-backward run was too significant for an overall benefit to be observed. My implementation therefore does not use gradient information at the line minimisation stage.

The main issue with the gradient calculation itself is memory usage. It is now necessary to store with each feature, as well as the usual properties (e.g. location and score), the u^f and u^b variables, each of which consists of an array of floating point numbers, the size of which is the number of different weights that are to be simultaneously optimised. If we wish to optimise weights for all evidence types for a simple, double-stranded model separately, then this requires an approximate additional 100 bytes per feature (16 feature types, 4 segment types, 5 length penalties), twice as much if we wish to use double-precision floating point arithmetic. Since a 1 Megabase sequence typically produces of the order of half a million features, this gives 50 Megabytes for the gradient variables alone!

The situation is never this bad in practice. Tying the forward and reverse versions of the features and segments together effectively halves the storage required, and I would expect this to be done for all sensible applications. In addition, it would be normal to tie together the weights for related feature types on the same strand, such as the six different splice sites (three donor, three acceptor).

4.7 A comparison with other methods

In this section I contrast the optimisation scheme described above with some of the ways in which parameters are estimated for other gene prediction systems. There are some elements of the scheme that are similar to methods used by exon-assembly predictors to incorporate scores from several distinct signal and content-sensors into a single exon score. It is also true that the procedure is probabilistic in flavour and has much in common with certain techniques used in the estimation of parameters for Hidden Markov models.

I present the comparison therefore in two parts. In the first part, I briefly examine some of the ways in which evidence of several different types is accommodated and weighted in some other gene prediction systems, relating them back to the GAZE strategy. The second part is more theoretical and discusses how the GAZE approach relates to various techniques used in the field of HMM parameter estimation.

4.7.1 Other methods based on weighted evidence

In many gene prediction systems, a variety of signal and content sensors are employed for the recognition of individual exons, which are then assembled into gene structures by dynamic programming. The specific way in which weights are chosen for the exon features varies. In FGENES for example [104], linear discriminant analysis (see chapter 1) is used to find a linear, weighted combination of a series of exon quality measures that provides maximum discrimination between true and false exons [103]. The exon likelihoods reported by the method are then used as the basis for a dynamic programming algorithm which identifies the highest scoring exon assembly. GRAIL [113][117] on the other hand uses a neural network to combine the scores from several signal and content sensors, trained to discriminate between true and false individual exons. The result gives rise to a procedure for identifying a set of candidate exons, which are assembled by a separate, dynamic programming procedure called GAP [118].

The weight optimisation technique presented in this chapter differs from these

methods by the fact that it considers whole gene structures[3]. I focus now on two methods that are similar to GAZE strategy in this respect.

GeneParser

The GAZE method is a fully implemented extension of the unimplemented framework proposed by Stormo and Haussler [109], generalised to account for a gene structure model that is not known in advance. The most comparable previously implemented work is the GENEPARSER system of Snyder and Stormo [101] [102]. The final gene prediction program is identical to the one proposed by Stormo and Haussler (section 4.2.1), but the difference is how optimal weights for various types of evidence are obtained.

Like GRAIL, GENEPARSER uses a neural network to combine the scores from several signal and content sensors. Unlike GRAIL however, the dynamic programming procedure to obtain the optimal parse is included in the neural network training regime. This allows for it to be trained not on individual exons but on complete gene structures. An iterative procedure is used, where the input to the network at each state is the difference (for each type of evidence) between the score of the *optimal* parse and the *correct* parse using the weights obtained in the last stage. A set of weights is thus obtained which maximises the number of training sequences for which the optimal parse is the correct one.

This approach is distinct from the maximum likelihood and maximum feature discrimination methods for GAZE, where the parameters are estimated with respect to the correct gene structure, or individual features belonging to it; the GENEPARSER method focuses directly on the relationship between the correct structure and the *highest scoring* structure. It could be argued that this makes more sense because it relates directly to the way in which the program will be used for prediction after

[3]This is true of both the ML and MFD methods; the latter is a function of feature *posterior* probabilities which, as explained in chapter 2, can be interpreted as a measure of how well the feature can be accommodated into a high-scoring complete gene structure.

training.

EuGene

EUGENE [99] is a program similar to GAZE in the way that it is tailored for integrating the output of external programs predicting gene features. It models the sequence as a directed acyclic graph (DAG) with the nodes organised into rows, or "tracks". Each track corresponds to a region of functional class and comprises a node for each base and edges joining the nodes of adjacent bases. The graph is then decorated with edges between tracks, depending on the results of the external feature-prediction programs. For example, if a program of choice reports donor splice site at position $(x, x+1)$, then an edge is added between node x of the "exon" track and node $x+1$ of the "intron" track. Once this is done, any path from the "source" node to the "sink" through the graph corresponds to a gene structure. Scores are added to the edges of the graph and the predicted gene structure is that corresponding to the "shortest" path through the graph[4].

What makes EUGENE particularly interesting from the perspective of the work presented here is the way in which the scores reported by the external programs are weighted. It is assumed by the authors that the scores c_i reported by a program predicting evidence type i are numbers between zero and one, and to transform such scores into a penalty they use the function $-\log(ac_i^b)$. The constants a and b are obtained by maximising the percentage of correct predictions in a set of single gene *A.thaliana* sequences. The optimisation is performed first by a simple genetic algorithm [50], and then by sampling the local parameter space. The procedure is extremely computationally intensive and is not guaranteed to find either a global or local maximum. The authors justify it however as a method that identifies sets of parameters that work well. A system with a similar architecture is DAGGER [25], which combines evidence using a complex function with two free variables for each type of (in their system fixed) evidence. Optimal values for the variables are found by

[4]The scores are therefore penalties, with a high score corresponding to an unlikely feature.

the downhill simplex function optimisation method [81] which does not make use of gradient information and is therefore be inefficient for multi-dimensional functions. The authors also give very little justification for the form of the objective function except that it works well.

It is interesting to note that although these methods have weaknesses, they go one step beyond the GAZE method in associating *two* parameters with each piece of evidence, namely a multiplication factor and an additive constant (analogous to bias terms in neural networks). This is however achievable in GAZE under certain circumstances by using a fixed, length-independent penalty function to act as the bias term for the transformation of a feature score (see for example section 5.5.1 describing the introduction of transcription start site predictions). Care must be taken to apply this technique in general as there would be indeterminacy between some sort of additive constants, for example, those for "intron" and "exons" regions.

The general approach of associating each model element with two parameters would double the number of free variables in the system, putting a strain on the time an memory usage of the training procedure, as well as making it much more vulnerable to over-fitting. For this this reason, I have not examined the possibility further. It is however a natural extension to this work to extend the gradient calculation to account for additive constants.

4.7.2 Hidden Markov model methods

In order to place a probabilistic interpretation upon the weight-optimisation procedure presented in this chapter, it is instructive to look to the way in which the parameters of methods based on Hidden Markov models are estimated.

The basic set of parameters for a HMM gene predictor are the state emission and transition probabilities. The standard approach to obtaining these parameters is maximum likelihood, whereby the probability of the observations (in this case a set of training sequences) is maximised:

$$\max_{\mathbf{w}} \sum_{k=1}^{K} \log \mathrm{P}(S_k|\mathbf{w})$$

I use $\mathbf{w}$ to denote the complete set of model parameters, to be consistent with the notation earlier. How this function is optimised depends on whether the training sequences come complete with annotated gene structures or not.

Supervised estimation

If each base of the training sequence is labelled according to its role in the gene structure ("intron", "exon", "UTR" etc.), and there is a one-to-one correspondence between the states of the model and these labels then a "correct" state path can be inferred directly from the labelled training sequence. The ML approach therefore is to calculate values for each transition and emission probability based on their frequency of occurrence in the correct paths of the training sequences[5]. This in fact corresponds to maximising the joint probability of the sequence and the correct path ϕ^*:

$$\mathbf{w}^{\mathrm{ML}} = \operatorname*{argmax}_{\mathbf{w}} \sum_{k=1}^{K} \log \mathrm{P}(S_k, \phi^*|\mathbf{w})$$

The same applies to the Generalised HMM architecture used by many of the most successful gene prediction programs, for example GENSCAN [21], GENIE [72], FGENESH [96] and developmental versions of GENEMARK.HMM [76]. In these models, it is often the case each state corresponds to a particular functional class (i.e. a single label). Sub-models can therefore be estimated for each component individually, based on only the corresponding regions from the training sequences. The sub-models are then combined with transition probabilities based again on counts in the paths inferred from the training sequences. The GHMM formalism also allows for arbitrary probability distributions over the durations of each state, although as

[5] With pseudo-counts being added when data available for the estimation of the parameters is sparse.

explained in chapter 1 for practical reasons this is usually only exploited for certain state types. State duration probability distributions can be derived directly from histograms compiled from observed data.

Unsupervised estimation

It is sometimes not possible to infer a correct state path from a training sequence. This could be because the sequence does not have an annotated gene structure, or because there is many-to-one relationship between states and annotation labels, meaning that there is more than one state path that could have given rise to the annotated gene structure. A general approach in this case is therefore to maximise the unconditional probability of the training sequence(s), summing over all paths (or if there is partial information, summing over all paths consistent with this information; see later). There are various iterative techniques for maximising the likelihood in this case, based on the fact that the log probability of a training sequence can be expressed as a sum over all state paths, the partition function:

$$\log \mathrm{P}(S|\mathbf{w}) = \log \sum_{\phi} \mathrm{P}(S, \phi|\mathbf{w})$$

I showed above how the gradient of this function can be calculated in the context of the GAZE scoring function, and for HMMs the result is similar [67]:

$$\frac{\partial}{\partial w} \log \mathrm{P}(S|\mathbf{w}) = \frac{n_k(S)}{w}$$

where $n_k(s)$ is the "expected" number of uses of the kth parameter in generating the training sequences, i.e. the weighted average over all possible paths. This can be calculated by the standard forward-backward algorithm for HMMs [90] [36]. A gradient descent method such as that presented earlier can therefore be used to maximise the likelihood. However, the more efficient *Baum-Welch* procedure [5] is often used. The algorithm works on the principle that by estimating new parameter values from "expected" counts above, the log-likelihood of the training sequences is *necessarily* increased.

An example of the use of the Baum-Welch method can be found in one of the first applications of HMMs to gene prediction [70]. The authors use the algorithm to estimate a profile-based HMM for two distinct types of intergenic regions in the genome of *E. Coli*. Elements of the resulting model were observed that were biologically meaningful, validating the method.

A combined approach: HMMGene

HMMGENE is a gene prediction program based on the notion of a *Class* Hidden Markov model (CHMM), where each state is associated with a label (see chapter 1). This framework allows for the identification of the most probable *labelling* (and hence most likely gene structure) in addition to the most probable state path.

The estimation of the parameters for HMMGENE begins with a set of labelled training sequences, but it is not possible to estimate maximum likelihood parameters from counts directly. Although HMMGENE implements a restricted version of the CHMM framework whereby each state emits a single, constant label, it is still true that a given label could have been generated by more than one state.

The natural approach to this problem is to use a modified version of a Baum-Welch algorithm where, during the forward-backward calculation, only valid path through the model are allowed. In the context of labelled sequences, a valid path is one in which the state labels agree with the sequence labels. This maximises the joint probability of the sequence and the labelling L:

$$\mathbf{w}^{\mathrm{ML}} = \underset{\mathbf{w}}{\operatorname{argmax}} \log \mathrm{P}(S, L|\mathbf{w})$$

This is the function that is optimised in the training of the VEIL program, which also uses the CHMM framework [56]. The problem with this technique is that it can often give rise to parameter values that are poor for prediction; although the probability of emitting a series of "Exon" labels for an exonic region of the training sequence is made as high as possible, the parameters so obtained can make the probability of emitting a series of "Intron" or "Intergenic" labels with the sequence even higher.

HMMGENE uses a technique called *Conditional* Maximum Likelihood (CML), to address this problem. The idea is to optimise the probability of the correct *labelling*, given the sequence:

$$\begin{aligned} \mathbf{w}^{\mathrm{CML}} &= \underset{\mathbf{w}}{\operatorname{argmax}} \log \mathrm{P}(S|L,\mathbf{w}) \\ &= \underset{\mathbf{w}}{\operatorname{argmax}} \left[\log \mathrm{P}(S,L|\mathbf{w}) - \log \mathrm{P}(S|\mathbf{w})\right] \end{aligned} \tag{4.24}$$

where $\mathrm{P}(S|\mathbf{w})$ is the sum over all state paths and labellings. This can be computed with the standard forward algorithm (i.e. ignoring the labels). Since there is a one-to-one correspondence between labellings and gene structures, maximising it corresponds to maximising the probability of the annotated gene structure.

The effect of CML is that the relative difference between the probabilities of consistent state paths (i.e. those where the labelling of the states agrees with the labelling of the training sequence) and inconsistent state paths (i.e. where the state labelling does not agree with the labelling of the training sequences) is maximised. This ideally results in the probability of invalid paths being very low, meaning that *all* paths through the model (with significant probability) will give rise to the correct gene structure. Since the aim of CML is to arrive at a set of parameters that provides maximum discrimination between consistent and inconsistent state paths (with respect to the labelling of the training sequences), the technique is also known as *discriminative estimation* [36].

Relationship to the GAZE approach

The estimation problem in GAZE is simplified with respect to the above in that the parameters of signal and content recognition sub-models are fixed. The problem is therefore to find a way to ascribe relative importance to each element so that their resulting integration results in gene predictions that are as accurate as possible.

To see how my training method relates to HMM training, it is insightful to consider the GAZE gene structure probabilities as implicitly conditional upon the

underlying sequence (see section 2.5). Making the conditioning explicit, the ML objective function (equation 4.6) can therefore be written as:

$$\begin{aligned}
\mathbf{w}^{\mathrm{ML}} &= \underset{\mathbf{w}}{\operatorname{argmax}} \ln \mathrm{P}(\phi^* | S, \mathbf{w}) \\
&= \underset{\mathbf{w}}{\operatorname{argmax}} E(\phi^*, \mathbf{w}) - \ln \sum_{\phi \in \Phi} \mathrm{e}^{E(\phi, \mathbf{w})} \\
&= \underset{\mathbf{w}}{\operatorname{argmax}} \ln \mathrm{P}(\phi^*, S | \mathbf{w}) - \ln \mathrm{P}(S | \mathbf{w}) \qquad (4.25)
\end{aligned}$$

When using this framework it is assumed that a correct gene structure (in terms of a list of features) can be inferred directly from an annotated sequence. Since this means that there is no distinction between gene structures and labellings, the above might be interpreted as a kind of conditional maximum likelihood. This makes intuitive sense, since like CML, the above minimises the probability of all incorrect gene structures. The principle difference between the above and CML in the CHMM context is that the latter simultaneously estimates the transition and emission probabilities of the sub-models. In the context of GAZE, the sub-models are implicit in the features, hence their internal model parameters are not variable.

The MFD method goes one step beyond CML in attempting to discriminate between individual incorrect gene structures as well as between the single correct and all incorrect structures. The idea is that even in the case where the highest scoring gene structures are not correct, they at least use as many elements (features) of the correct structure as possible. This should intuitively give rise to sets of weights that are more accurate. Whether this turns out to be the case is examined in the following chapter.

To end, it is interesting to consider whether unsupervised learning methods are applicable to the estimation of optimal evidence weights in GAZE. Maximising the unconditional probability of the training sequences (i.e. the partition function) directly by classical methods is not possible because a GAZE score is not a real log-probability; there is no comparison to other sequences. It is likely therefore that increasing the weights will increase the partition function in an unbounded

way. The possibility of an indirect method however remains open. The Baum-Welch method is a special case of *Expectation Maximisation* [31] for maximum-likelihood-style estimation in the case where certain data (in this case the correct gene structures) are not known. Specifically, expectation maximisation is based upon the following calculation:

$$\mathbf{w}^{t+1} = \underset{\mathbf{w}}{\operatorname{argmax}} \sum_{\phi} \mathrm{P}(\phi|S, \mathbf{w}^t) \log \mathrm{P}(\phi, S|\mathbf{w})$$

It can be shown that under a wide range of conditions, this approach necessarily increases the likelihood of the model, and in many situations, the maximum is calculable directly without iteration. Such a case is that of HMMs and the Baum-Welch algorithm. The straightforward application of the EM method to HMMs arises due to their transparency in the way that once posterior path probabilities corresponding to expected gene structures are calculated by the forward-backward procedure, transition and emission probabilities that increase the objective function are obtained by simply summing the expected usages and normalising.

To apply EM to GAZE, we would have to use the Maximum Likelihood approach, weighting gene structure probabilities by their posterior probability in the previous iteration:

$$\begin{aligned} \mathbf{w}^{t+1} &= \underset{\mathbf{w}}{\operatorname{argmax}} \sum_{\phi} \mathrm{P}(\phi|\mathbf{w}^t)\mathrm{P}(\phi|\mathbf{w}) \\ &= \underset{\mathbf{w}}{\operatorname{argmax}} \frac{\sum_{\phi} \mathrm{e}^{E(\phi|\mathbf{w}^t)+E(\phi|\mathbf{w})}}{[\sum_{\phi} \mathrm{e}^{E(\phi|\mathbf{w}^t)}][\sum_{\phi} \mathrm{e}^{E(\phi|\mathbf{w})}]} \end{aligned}$$

It is not obvious how this quantity might be maximised analytically, as is possible with HMMs and the Baum-Welch algorithm. The standard numerical optimisation techniques however are applicable. To use conjugate gradient descent method in particular would the require the computation of the gradient of the above quantity with respect to the model element weights. Such an approach would be a natural extension to this work.

4.8 Optimising evidence weights for GAZE_EST

In the previous chapter, I described the design and refinement of a GAZE model for the prediction of genes in *C. elegans* sequences. In the final stage of this refinement, data derived from EST alignments were used to improve the accuracy of the system. However, the weights for the scores for these data were determined in a rather *ad hoc* way. The Maximal Feature Discrimination training method provides the opportunity to attack this problem in a more principled way. To end the chapter, I illustrate the use of MFD training by applying it to this specific problem. A more rigorous examination of the method in comparison with maximum likelihood is detailed in the following chapter, in the context of the development of a gene prediction system for vertebrate sequences.

4.8.1 Choice of parameters and optimisation method

The types of additional evidence in the GAZE_EST model (with respect to the GAZE_trans model) are described in section 3.6.2. They are four features ("transcript_start" and "transcript_stop", and reverse strand versions of these) and five segments (forward and reverse strand versions of "EST_match" and "EST_intron", as well as the "EST_span" segments which were introduced to prevent gene splitting). Tying the weights of corresponding forward and reverse strand elements together results in five parameters to optimise. However, since the transcription initiation and termination features have zero score (see section 3.6.2), any scaling applied to them will have no effect. In addition, the "EST_span" segments are designed to eliminate from consideration any candidate gene structures that make use of them; since the correct gene structure will not make use of them, any weight attached to their scores will be unbounded. It is appropriate therefore to optimse only two parameters.

For the work in chapter 3, the WormSeq dataset was used to assess the accuracy of various GAZE models. To be able to use this dataset for training as well as testing, it was split in half (adjusting the division point slightly to occur in the intergenic region between two genes). I refer to these two new sequences as *WormSeq-1* and

WormSeq-2. Splitting the dataset in this this way enables a twofold cross-validation, testing the parameters obtained by training on WormSeq-1 on WormSeq-2, and vice versa.

Of the two objective functions described earlier, only Maximal Feature Discrimination is applicable in this instance. The reason for this is that since only coding regions of the confirmed genes in WormSeq are known, the correctness of certain types of candidate feature (specifically *trans*-splice sites, and transcription and initiation sites) is not known during the training. However, MFD allows the declaration of a subset of feature types that should be ignored in the optimisation (see section 4.3.2).

4.8.2 Accuracy of the trained model

Table 4.1 shows the weights obtained from the MFD optimisation, and table 4.2 shows the accuracy resulting from using these weights. The most noticeable difference in the values compared with those chosen by hand in chapter 3 is that the weight for "EST_match" segments has doubled. However, this gives only marginal (if any) improvement in accuracy. It might be argued that this should not be too surprising, since the reason for choosing the original weights is that they seemed to give good accuracy.

One point that was not considered when choosing the weights in the previous chapter is that by introducing additional elements to the model, the weights of the existing model elements may no longer perform well. In this application, "EST_match" and "coding_seg" segments both contribute towards the score for candidate protein coding exons. It is therefore possible that the scores for "coding_seg" segments have to be down-weighted to compensate for the addition of "EST_match" segments. Tables 4.1 and 4.2 also show the result of simultaneously optimising weights for the "EST_match", "EST_intron" and "coding_seg" segments (the entry labelled GAZE_EST$^{\mathrm{MFD}-3}$). The optimal value for the "coding_seg" weight is 0.4 (compared with 1.0 used in chapter 3, and the new set of weights gives significant

	EST_match	EST_intron	coding_seg
MFD-2^1	0.02	0.05	-
MFD-2^2	0.02	0.06	-
MFD-3^1	0.02	0.05	0.42
MFD-3^2	0.02	0.06	0.41
Chapter 3	0.01	0.05	1.00

Table 4.1: Evidence scoring weights determined by Maximal Feature Discrimination. In addition to the weights used in chapter 3 (bottom row), the final weights for the relevant elements are shown for both two-parameter (MFD-2) and three-parameter (MFD-3) training runs described in the text, and for each training sequence WormSeq-1 and WormSeq-2 (indicated by superscript 1 and 2 respectively).

(a)

Exon level	WormSeq-1					WormSeq-2				
	Sn	Sp	Av	ME	WE	Sn	Sp	Av	ME	WE
GAZE_EST$^{\mathrm{MFD-2}}$	0.89	0.83	0.86	0.02	0.09	0.90	0.85	0.87	0.02	0.09
GAZE_EST$^{\mathrm{MFD-3}}$	0.88	0.88	0.88	0.06	0.06	0.88	0.89	0.89	0.06	0.05
GAZE_EST	0.89	0.83	0.86	0.02	0.10	0.90	0.84	0.87	0.02	0.09

(b)

Gene level	WormSeq-1					WormSeq-2				
	Sn	Sp	Av	MG	WG	Sn	Sp	Av	MG	WG
GAZE_EST$^{\mathrm{MFD-2}}$	0.56	0.50	0.53	0.01	0.01	0.62	0.58	0.60	0.01	0.07
GAZE_EST$^{\mathrm{MFD-3}}$	0.59	0.58	0.58	0.04	0.07	0.65	0.64	0.65	0.03	0.04
GAZE_EST	0.54	0.49	0.52	0.01	0.11	0.61	0.58	0.60	0.01	0.07

Table 4.2: Accuracy of the GAZE_EST configuration after training with Maximal Feature Discrimination. For the trained models, the results for WormSeq-1 were obtained by training on WormSeq-2, and vice versa. The first rows are the result of training two parameters (weights for "EST_match" and "EST_intron" segments), and the second rows a further parameter in addition (a weight for "coding_seg" segments). The bottom row represents the accuracy of the weights used in chapter 3 when the model is applied separately to WormSeq-1 and WormSeq-2. The accuracy measures are explained in section 1.4.1.

improvement in accuracy (for example a 5-6% increase at the gene level average).

This simple application shows Maximal Feature Discrimination to be a good method for identifying sets of weights that give rise to accurate gene predictions. The effectiveness of the method is examined in more detail in the next chapter, where amongst other things it is compared to the more traditional maximum likelihood method.