Lecture 6

A.L. Yuille

February 5, 2012

Introduction

This lecture first describes exponential distributions and their relation to sufficient statistics and the maximum entropy principle. Secondly, we describe maximum likelihood for estimating the model parameters and show that for exponential distributions this is a convex minimization problem which corresponds to finding the parameter values such that the expected value of the statistics is equal to their observed values. Thirdly, we describe algorithms for performing maximum likelihood learning. Fourthly, we illustrate this for models using image statistics and show that MRF models can be learnt from image statistics. Fifthly, we describe how to use model selection to search through a set of different models which use different statistics.

1 Exponential Distributions

Almost all distributions (e.g. Gaussian, Poisson, etc.,) can be expressed as exponential models of form:

$$P(\vec{x}|\vec{\lambda}) = \frac{1}{Z[\vec{\lambda}]} \exp\{\vec{\lambda} \cdot \vec{\phi}(\vec{x})\}, \tag{1}$$

where $\vec{\phi}(\vec{x})$ denotes the *statistics* and $\vec{\lambda}$ the *parameters*. The normalization term $Z[\vec{\lambda}] = \sum_{\vec{x}} \exp\{\vec{\lambda} \cdot \vec{\phi}(\vec{x})\}$ (replace $\sum$ by $\int$ if $\vec{x}$ is continuous valued).

For example, the Gaussian distribution in one-dimensions has statistics $\vec{\phi}(x) = (x, x^2)$. This gives a distribution $P(x|\vec{\lambda}) = \frac{1}{Z[\vec{\lambda}]} \exp\{\lambda_1 x + \lambda_2 x^2\}$. This relates to the usual formulation $\frac{1}{\sqrt{2\pi}\sigma} \exp\{-(x-\mu)^2/(2\sigma^2)\}$ by the relations $\lambda_1 = \mu/\sigma^2$, $\lambda_2 = -(1/2)\sigma^2$, and $Z[\vec{\lambda}] = \sqrt{2\pi}\sigma \exp\{\mu^2/(2\sigma^2)\}$.

The $\vec{\phi}(\vec{x})$ are the *sufficient statistics*. They are all you know about the data. If you have data $(\vec{x}_1, ..., \vec{x}_n)$, then you only store samples $(1/n)\sum_{i=1}^n \vec{\phi}(\vec{x}_i)$ – (e.g. for a Gaussian $(1/n)(\sum_{i=1}^n x_i, \sum_{i=1}^n x_i^2)$).

Another way to obtain exponential distributions is by using the maximum entropy principle (Jaynes). Recall that the entropy of a distribution $P(\vec{x})$ is $H(P) = -\sum_{\vec{x}} P(\vec{x}) \log P(\vec{x})$ (for discrete distributions) or $\int d\vec{x} P(\vec{x}) \log P(\vec{x})$ (the differential entropy for continuous distributions). By Shannon's definition, the entropy is the amount of information you obtain from a sample $\vec{x}$ from the distribution. In the discrete case, the distribution with maximum entropy is the uniform distribution $P(\vec{x}) = 1/N$, where N is the number of possible states $\vec{x}$. Hence a considerable amount of information is obtained from a sample (because we do not know in advance what the sample will be). By contrast, the distribution with minimum entropy is of form $P(\vec{x}_0) = 1$ for one value $\vec{x}_0$ and $P(\vec{x}) = 0$ for $\vec{x} \neq \vec{x}_0$. It has entropy 0 since no information is obtained by sampling from $P(\vec{x})$ because we know the result must be $\vec{x}_0$.

The maximum entropy principle proposes that you select the distribution $P(\vec{x})$ whose expected statistics $\sum_{\vec{x}} P(\vec{x})$ take a fixed value $\vec{\psi}$. This selects the distribution $P(\vec{x})$ that maximizes:

$$-\sum_{\vec{x}} P(\vec{x}) \log P(\vec{x}) + \mu\{\sum_{\vec{x}} P(\vec{x})\vec{\phi}(\vec{x}) - 1\} + \vec{\lambda} \cdot \{\sum_{\vec{x}} P(\vec{x})\vec{\phi}(\vec{x}) - \vec{\psi}\}. \tag{2}$$

Here μ and $\vec{\lambda}$ are lagrange multipliers used to impose the normalization and expected statistics constraints. This can be maximized to yield:

$$P(\vec{x}) = \frac{1}{Z[\vec{\lambda}]} \exp\{\vec{\lambda} \cdot \vec{\phi}(\vec{x})\}, \tag{3}$$

where the parameter $\vec{\lambda}$ is chosen so that $\sum_{\vec{x}} P(\vec{x})\vec{\phi}(\vec{x}) = \vec{\psi}$.

2 Maximum Likelihood

Suppose we have a set of independent, identically, distributed (i.i.d.) data $D = (\vec{x}_1, ..., \vec{x}_n)$. Maximum likelihood estimates the parameters $\vec{\lambda}$ by maximizing the probability of the data:

$$P(D|\vec{\lambda}) = \prod_{i=1}^{n} P(\vec{x}_i|\vec{\lambda}). \tag{4}$$

This can be re-expressed as estimating $\vec{\lambda}$ by

$$\vec{\lambda}^* = \arg\min_{\vec{\lambda}} \sum_{i=1}^{n} \log P(\vec{x}_i|\vec{\lambda}). \tag{5}$$

Note: the standard maximum likelihood (ML) derivation assumes that the data is really generated by the distribution $P(\vec{x}|\vec{\lambda})$ for some value of $\vec{\lambda}$. But there is no guarantee that the data is really generated by this model. Instead we can derive the distribution of the data $P_D(\vec{x}) = \frac{1}{n}\sum_{i=1}^{n} \delta(\vec{x} - \vec{x}_i)$. Then the ML can be derived to minimize the Kullback-Leibler divergence $\sum_{\vec{x}} P_D(\vec{x}) \log \frac{P_D(\vec{x})}{P(\vec{x}|\vec{\lambda})}$ between the data distribution $P_D(.)$ and the model $P(.,.)$. This justifies ML as a way to find the best approximation of the data of form $P(\vec{x}|\vec{\lambda})$. Amari's information geometry gives a nice extension of these ideas.

The ML condition for exponential distribution reduces to minimizing the function:

$$F(\vec{\lambda}) = -\sum_{i=1}^{n} \vec{\lambda} \cdot \vec{\phi}(\vec{x}) + n \log Z[\vec{\lambda}]. \tag{6}$$

This is a convex function of $\vec{\lambda}$ (because the Hessian $\frac{\partial^2}{\partial \vec{\lambda}^2} \log Z[\vec{\lambda}]$ is positive semi-definite – as can be shown using the Cauchy-Schwartz inequality).

The first derivative of $\log Z[\vec{\lambda}]$ is the expected value of the statistics $\vec{\phi}(.)$:

$$\frac{\partial}{\partial \vec{\lambda}} \log Z[\vec{\lambda}] = \sum_{\vec{x}} \vec{\phi}(\vec{x}) P(\vec{x}|\vec{\lambda}),. \tag{7}$$

Hence the minimum of $F(\vec{\lambda})$ occurs at the value of $\vec{\lambda}$ such that the expected statistics of the model are equal to the statistics of the data:

$$\sum_{\vec{x}} \vec{\phi}(\vec{x}) P(\vec{x}|\vec{\lambda}) = \frac{1}{n} \sum_{i=1}^{n} \vec{\phi}(\vec{x}_i). \tag{8}$$

where we replace $\sum_{\vec{x}}$ by $\int d\vec{x}$ if $\vec{x}$ is a continuous variable.

For many well-known distributions – like Gaussian and Poisson – the solution $\vec{\lambda}^*$ can be computed analytically. This is because the expected statistics of the models can be computed analytically (i.e. we can compute the summation/integration of the terms on the left hand side of equation (**??**)). For example, for a one-dimensional Gaussian with $\vec{\phi}(x) = (x, x^2)$ we compute $\int dx \vec{\phi}(x) P(x|\mu, \sigma) = (\mu, \mu^2 + \sigma^2)$. Equating this with the data statistics $1/n(\sum_{i=1}^{n} x_i, \sum_{i=1}^{n} x_i^2)$ yields the standard estimates $\hat{\mu} = 1/n \sum_{i=1}^{n} x_i$ and $\hat{\sigma}^2 = 1/n \sum_{i=1}^{n} (x_i - \hat{\mu})^2$.

But for other distributions, and in particular, Markov Random Fields (MRFs), it is impossible to compute the expectation of the statistics analytically and hence we cannot solve equation (8) directly. Instead we need learning algorithms which we will describe in the next section.

3 Algorithms for Learning by Maximum Likelihood

The solution to ML is obtained by minimizing the convex function $F(\vec{\lambda})$. We describe two algorithms for doing this. Unfortunately both algorithms require an inner loop which computes the expectation of the statistics $\vec{\phi}(\vec{x})$ which can be problematic. We summarize the data statistics by $\vec{\psi} = \frac{1}{n}\sum_{i=1}^{n} \vec{\phi}(\vec{x}_i)$.

First, we can perform steepest descent on $F(\vec{\lambda})$, using equation (7) to compute the derivative of $F(.)$. This gives update equations:

$$\vec{\lambda}^{t+1} = \vec{\lambda}^{t} - \Delta\{\sum_{\vec{x}} \vec{\phi}(\vec{x})P(\vec{x}|\vec{\lambda}^{t}) - \vec{\psi}\}. \tag{9}$$

Secondly, we can generalized iterative scaling (GIS), to obtain discrete iterative update equations (these can be re-derived from CCCP) which give update equations:

$$\vec{\lambda}^{t+1} = \vec{\lambda}^{t} - \log \sum_{\vec{x}} \vec{\phi}(\vec{x})P(\vec{x}|\vec{\lambda}^{t}) + \log \vec{\psi}. \tag{10}$$

Here we use the convention that the logarithm of a vector is the vector of the logarithm of the components – i.e. $\log \vec{\psi} = (\log \psi_1, ..., \log \psi_m)$ where $\vec{\psi} = (\psi_1, ..., \psi_m)$.

The update equations for steepest descent and GIS are very similar. Both update the parameters $\vec{\lambda}$ by terms which depend on the difference between the data statistics $\vec{\psi}$ and the current model estimates $\sum_{\vec{x}} \vec{\phi}(\vec{x})P(\vec{x}|\vec{\lambda}^{t})$ and will converge to solutions of the ML equations in equation (8). The main difference is that GIS uses logarithms and does not require selecting a step size Δ.

But the difficulties of both algorithms – equations (9,10) – is that they require computing the expectation of the statistics – $\sum_{\vec{x}} \vec{\phi}(\vec{x})P(\vec{x}|\vec{\lambda}^{t})$. This cannot be done analytically (except for special cases like Gaussians) and requires an inner loop to compute it. But, even more seriously, for many distributions there are no known algorithms which can compute the expectation in polynomial time. This relates to the difficulty of doing inference for Markov Random Field (MRF) models – see previous lecture. Similarly, we can use approximate methods – variational/mean field theory, MCMC, belief propagation – to compute approximations to the expectations. If the probability models can be expressed in terms of distributions on graphs without closed loops then we can use dynamic programming to perform this (as will be discussed in later lectures).

These results show that there are close relations between learning and inference. By inference we mean estimating $\vec{x}^* = \arg\max_{\vec{x}} P(\vec{x}|\vec{\lambda}) = \arg\max_{\vec{x}} \vec{\lambda}\cdot\vec{x}$ (note: other authors use the word "inference" in a more general sense). Learning requires using the updates rules for steepest descent or GIS together with the inner loop which computes $\sum_{\vec{x}} \vec{\phi}(\vec{x})P(\vec{x}|\vec{\lambda})$. Hence in both cases we either have to sum/integrate or maximize quantities which are functions of $\vec{x}$. (Note: say that some machine learning algorithms reduce to doing maximization and not summation).

4 Learning MRFs: Weak Membranes and Image Derivative Statistics

Now we give an example by applying Maximum Likelihood to learn probability models of images. This requires selecting image statistics for the model (the next section discusses we can select between different statistics and how to combine them).

What statistics to use for images? First consider a real simple model that compute the histograms of the intensities of each pixels. This gives statistics $h(\vec{x}; z) = \frac{1}{n}\sum_{i=1}^{n} \delta(x_i - z)$ for an image $\vec{x} = (x_1, ..., x_n)$. This relates to the notation in our chapter by setting $\vec{h}(\vec{x})$ to be a 256-dimensional vector with components

$h(\vec{x};z)$, where z takes values in $\{0,1,...,255\}$ (i.e. the set of possible intensity values). This corresponds to an exponential distribution:

$$P(\vec{x}|\vec{\lambda}) = (1/Z(\vec{\lambda}))\exp\{\vec{\lambda}\cdot\vec{h}(\vec{x})\},\ P(\vec{x}|\vec{\lambda}) = (1/Z(\vec{\lambda}))\exp\{\sum_{z=0}^{255}\lambda(z)\frac{1}{n}\sum_{i=1}^{n}\delta(x_i - z)\},\ P(\vec{x}|\vec{\lambda}) = (1/Z(\vec{\lambda}))\exp\{\sum_{i=1}^{n}(1/n)\lambda(x_i)\},$$

$$P(\vec{x}|\vec{\lambda}) = \prod_{i=1}^{n}P(x_i|\vec{\lambda}) \text{ with } P(x_i|\vec{\lambda}) = \frac{1}{Z_i(\vec{\lambda})}\exp\{(1/n)\lambda(x_i)\} \text{ with } Z_i(\vec{\lambda}) = \sum_{x_i=0}^{255}\exp\{(1/n)\lambda(x_i)\}.$$

Note that the form of the statistics – histograms of the intensities for the image – leads to a probability distribution which is factorized so that the intensity at each pixel is generated independently.

We train the model on a set of images $\{\vec{x}^\mu : \mu \in \Lambda\}$. We compute the data statistics to be $\hat{h}(z) = (1/|\Lambda|)\sum_{\mu\in\Lambda}h(\vec{x}^\mu, z)$. Then ML estimation reduces to solving the equivalent equations:

$$\sum_{\vec{x}} P(\vec{x}|\vec{\lambda})h(\vec{x}(z) = \hat{h}(z)\ \forall\ z,\quad P(x_i) = \hat{h}(x_i). \tag{12}$$

For these statistics the ML problem reduces to using a factorized probability model $P(\vec{x}) = \prod_{i=1}^{n}P(x_i)$ and learning the components $p(x_i)$ directly from the data statistics. But this is an unrepresentative case. Moreover, the histograms of image intensities differs a lot between different images, so there are not likely to yield a good model for images.

Now, instead let us consider the statistics of the image differences – $I_i - I_j$ where i and j are neighboring pixels. These correspond to the statistics of the first order derivatives of the images (after discretization). It has been shown (references) that these statistics of the derivatives are remarkably similar between images and hence are capturing information about the nature of images.

We do this in one-dimensions to simplify the analysis. In this case the histograms of first-order derivatives/differences are represented as follows:

$$h(\vec{x};z) = \frac{1}{n}\sum_{i=1}^{n}\delta_{x_{i+1}-x_i,z}, \tag{13}$$

where $\delta_{a,b}$ is the delta function (i.e., $\delta_{a,b} = 1$, if $a = b$ and $\delta_{a,b} = 0$, if $a \neq b$). (Note: this represents every possible value for the difference from -255 to 255 – in practice we use a smaller number of coarse bins).

The distribution is given by:

$$\begin{aligned} P(\vec{x}|\lambda) &= \frac{1}{Z[\lambda]}\exp\{\sum_z \lambda(z)h(\vec{x};z)\},\\ &= \frac{1}{Z[\lambda]}\exp\{\sum_z\sum_{i=1}^{n}(1/n)\lambda(z)\delta_{x_{i+1}-x_i,z}\},\\ &= \frac{1}{Z[\lambda]}\exp\{(1/n)\sum_{i=1}^{n}\lambda(x_{i+1}-x_i)\}. \end{aligned} \tag{14}$$

This derivative shows that the choice of statistics – the histogram of the first order derivatives/differences – leads to a Markov Random Field distribution. The size of the neighborhood for the statistics determines the neighborhood for the MRF. See graph figures. Observe that using the histogram of the statistics of the second order derivative d^2/dx^2 would give a model with interactions to pairs of neighbors. *Hence the graph representation of the distribution is determined by the statistics.*

To determine the value of the potentials $\vec{\lambda}(.)$ we have to observe the statistics of the training image dataset $h(z)$ and solve for $\vec{\lambda}^*$ so that $\sum_{\vec{x}} h(\vec{x};z)P(\vec{x}|\vec{\lambda}^*) = h(z)$. This cannot be done analytically (unlike

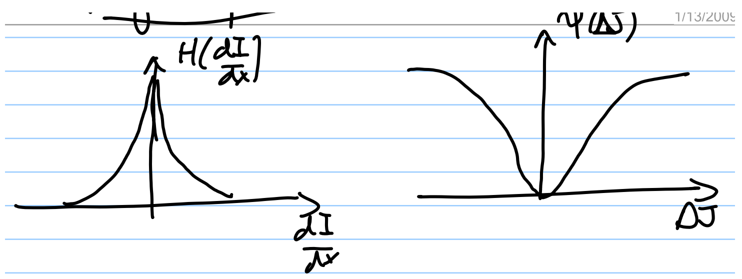

Figure 1: Left Panel: the typically histogram observed for derivative filters. Right panel: the corresponding potential obtained by maximum likelihood learning.

the previous example). Instead we must follow the procedure defined in the previous section. This requires taking expectations of the statistics – in this simple example we can use dynamic programming to exploit the one-dimensional structure and compute the expectations in polynomial time. If we extend the model to a two-dimensional image then we can use MCMC or some of the approximate algorithms (Zhu and Mumford used MCMC to perform this.).

What values of $\vec{\lambda}$ do we expect to get? Note that the statistics of the image derivatives show that images are locally smooth (have low derivatives) but can have large discontinuities. These, of course, are the same reasons used to justify the weak smoothness models (Mumford and Shah, Rudin, Osher, and Fatemi, Geman and Geman, Blake and Zisserman). Hence we expect that learning will yield a type of weak smoothness model.

This is what the learning gives. It outputs a potential function $\vec{\lambda}$ (with a sign change) which takes small values if $|I_i - I_{i+1}|$ is small and which reaches an asymptotic limit for larger $|I_i - I_{i+1}|$. This is consistent with a model like Mumford-Shah (also Geman and Geman/ Blake and Zisserman) which attempts to smooth small fluctuations in the image gradient but which puts in breaks/edges if the edge gradient becomes too large. See figure (1).

Small point: advanced topic. For this example, there is a simple approximate (Coughlan and Yuille) which can be used to determine an analytic form for the potentials $\lambda(.)$ in terms of the observed statistics $h(.)$. This approximation requires that the distribution of derivative filters at different pixels in the image are independent, *when the distribution of the image is independent random noise* (per pixel). This is only an approximation (because even for random noise images there is some dependence of the derivatives at neighboring pixels in the image). Assuming this approximation, it can be shown that $\lambda(z) = \log \frac{h(z)}{a(z)}$ where $a(z)$ is the histogram of the derivative filter responses on random noise images.

At first sight, this argument seems to validate the weak membrane model by showing that it corresponds to the statistics of natural images. But this assumes that we only care about the first order derivative statistics of images. Several authors have shown that the statistics of higher order derivatives on images also have similar histograms which are also similar from image to image (Lee and Mumford, Green). Hence the weak membrane model is ignoring these higher order statistics and so is far too local a model.

5 Selecting Features: Sparsity and Model Selection

Note: not covered in lectures due to time constraints.

What if we do not know the features $\phi(x)$? One strategy is to specify a *dictionary* of features $\{\vec{\phi}^a(.) : a \in A\}$ and assign parameters $\{\vec{\lambda}_a\}$ to all of them:

$$P(\vec{x}|\{\vec{\lambda}_a\}) = \frac{1}{Z[\{\vec{\lambda}_a\}]} \exp\{\sum_a \vec{\lambda}_a \cdot \vec{\phi}_a(\vec{x})\}. \tag{15}$$

We can put a sparsity prior on the parameters $P(\{\vec{\lambda}_a\}) = \frac{1}{Z} \exp\{-K \sum_a |\vec{\lambda}_a|\}$.

Then we perform Maximum a Posteriori (MAP) estimation – estimate $\{\vec{\lambda}_a\}$ from $P(\{\vec{\lambda}_a\}|D) \propto P(D|\{\vec{\lambda}_a\})$ where $D = \{\vec{x}_\mu\}$ is the training data. The sparsity prior will encourage many of the $\vec{\lambda}_a$ to be zero. This means that the many of the features will not be used. Hence sparsity allows us to *select* which features to use as well as to weight (i.e. assign non-zero $\vec{\lambda}_a$) to those that we do select. (A similar strategy relates to the AdaBoost learning algorithm, as we will discuss in a later lecture).

But this strategy becomes extremely computationally demanding if the number of features in the dictionary is large. An alternative related strategy is more greedy (Della Pietra et al, Zhu et al) and involves selecting features to use in order. This requires some knowledge of model selection.

Consider two models: $P_1(\vec{x}|\vec{\lambda}_1) = \frac{1}{Z[\vec{\lambda}_1]} \exp\{\vec{\lambda}_1 \cdot \vec{\phi}_1(\vec{x})\}$ and $P_2(\vec{x}|\vec{\lambda}_2) = \frac{1}{Z[\vec{\lambda}_2]} \exp\{\vec{\lambda}_2 \cdot \vec{\phi}_2(\vec{x})\}$.

For each model we can estimate the best values of the parameters $\hat{\vec{\lambda}}_1, \hat{\vec{\lambda}}_2$ by performing ML estimation from the data $D = \{\vec{x}_\mu\}$. We can then compute $P_1(D|\hat{\vec{\lambda}}_1)$ and $P_2(D|\hat{\vec{\lambda}}_2)$ – recall that these relate to the entropies of these models (previous lectures).

One form of model selection is to select the model for which $P_i(D|\hat{\vec{\lambda}}_i)$ is largest – i.e. which has highest probability of generating the data, or equivalently, which has lowest entropy for the data (and hence describes it better). This method may need to be augmented by a penalty term in case one model has more parameters than the other (e.g., criteria like AIC and BIC). Arguable a letter method of model selection to to put priors $P_1(\vec{\lambda}_1), P_2(\vec{\lambda}_2)$ on the $\vec{\lambda}$ parameters and then compare $P_1(D) = \sum_{\vec{\lambda}_1} P_1(\vec{\lambda}_1) P_1(D|\hat{\vec{\lambda}}_1)$ with $P_2(D) = \sum_{\vec{\lambda}_2} P_2(\vec{\lambda}_2) P_2(D|\hat{\vec{\lambda}}_2)$. It can be shown (see Mackay) that this automatically gives an "Occam factor" which automatically corrects for the complexity of the models. The intuition is that the probability of all datasets must sum to one for each model – i.e. $\sum_D P_1(D) = 1$. Hence if a model has many free parameters (hence has high complexity) it can fit many datasets and some must assign some probability to each of them, thereby preventing it from assigning a high probability to any particular dataset. But a model with a smaller number of parameters will typically only have good fits to a smaller number of datasets, and hence can assign higher probabilities to them. Hence this form of model selection favors models which are specific to the dataset. (Note: the complexity of the model is complex – it doesn't simply correspond to the number of free parameters of the model, e.g., the dimension of the vector $\vec{\lambda}_1$). In practice, however, summing/integrating over the model parameters $\vec{\lambda}$ is often impractical– in some case, it can be approximated using methods like Laplace, which gives a factor for penalizing the model.

Using model selection we can select between a large number of models each of which uses a single features from the dictionary. This gives a model with statistic $\vec{\phi}_{a1}(\vec{x})$ with parameter $\vec{\lambda}_{a1}(\vec{x})$. Next we grow the model by adding an extra feature $\vec{\phi}_i(\vec{x}))$ – giving a new class of models of form $P(\vec{x}|\vec{\lambda}_{a1}, \vec{\lambda}_i) = 1/Z[\vec{\lambda}_{a1}, \vec{\lambda}_i] \exp\{\vec{\lambda}_{a1} \cdot \vec{\phi}_{a1}(\vec{x}) + \vec{\lambda}_i \cdot \vec{\phi}_i(\vec{x})\}$. For each i we estimate the best value of the parameter $\vec{\lambda}_i$ (keeping the first set $\vec{\lambda}_{a1}$ fixed). We perform model selection to select the best feature as before. We proceed to add new features in this greedy manner.

This form of feature pursuit was first developed by Delle Pietra et al. for non-vision applications. Zhu, Wu, and Mumford applied it to vision and called it minimax entropy. This uses the maximum entropy principle to obtain exponential models from the statistics. It uses the minimal entropy principle to perform model selection (recall that the probability of the data $P(D|\hat{\vec{\lambda}})$ relates to the entropy of the distribution – selecting the model with highest probability of generating the data is equivalent to selecting the model with lowest entropy).