Inference

Iason Myttas[1], Andreas Hadjiantoni[2] and Panagiotis Dimitriou[3]

[1] *candidate number (97214)*
[2] *candidate number (35033)*
[3] *candidate number (97272)*

December 7, 2018

In this paper we examine different methods to approximate computationally and analytically intractable distributions. In particular, we attempt to denoise a binary valued image which was intentionally altered using Gaussian or binary noise. As all of the methods are exceptionally effective on images changed using the former type of noise, we will focus on the latter. We evaluate the effectiveness of each method by comparing the denoised image with the original using a mean squared error approach. We evaluate their efficiency by counting the number of iterations through the image they need until they terminate. The parameters η and β of each algorithm have been optimized by trial and error. We extend the use of those algorithms to include image segmentation (in a background and foreground sense) and examine Variational-auto encoders and their relationship to variational methods in the last part.

1 Approximate Inference

1.1 Iterative Conditional Modes

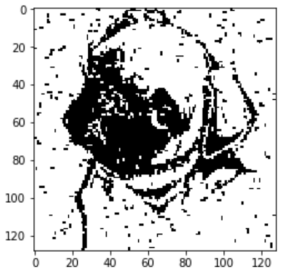
1.1.1 Question 1

The ICM implementation is the simplest algorithm that can infer the latent variables. Before we start with the implementation we had to decide on a representation for our prior and likelihood. Since we are dealing with a markov random field, the markov blanket for each pixel only involves its neighbors. We therefore decided that as a prior we suppose that the output of each pixel should depend on the values of its neighbors. For our likelihood, we reasoned that the correct function to be used is a negative distance function, so that when each x_i is similar to the y_i value observed, it should be more likely that x_i would retain its current value. We therefore use the notion of "negative exponential distance" to capture similarity between pixels. Otherwise, it will be encouraged to change from white to black and vice versa. To accomplish this we scaled the y values before the computations. We also introduced two parameters, β (beta) and eta (ita), that weigh the importance of the prior and the likelihood. It should be noted that the choice of these parameters is arbitrary and if we choose to work with the logs of the prior and likelihood, their effect will be exponentially higher. ICM works extremely well with Gaussian noises given highly weighted likelihoods, while in the salt and pepper case, the higher the value of η, the more clear the features of the latent representation are but more of the original noise remains, since the impact of the neighbors is shadowed. It is generally easier to deal with Gaussian noise in most of these models because the changes to the affected pixels are minor and therefore our negative distance function does not return high values. This means that the observed (possibly flawed) values don't have such a high impact on the latent variables even when they disagree. This explains why figure 2 is nearly identical to the original one. On the other hand, ICM is not as effective in the salt and pepper noise type, as it is visible in Figure 1.

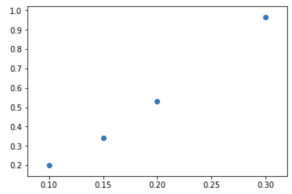
1.2 Stochastic Inference

1.2.1 Question 2

The Gibbs sampling algorithm is a Markov Chain Monte Carlo Algorithm. This means that it uses its previous state to stochastically build the new one. In particular, it samples from one dimension at a time, since the problem of sampling from all of the dimensions is intractable. Every time a sample is taken, the probability



(a) *Denoised salt and pepper image with ICM (β=2, η=0.8).*



(b) *Mean Squared Error (y-axis) of denoised images with ascending percentage noise levels (x-axis) with ICM.*

Figure 1

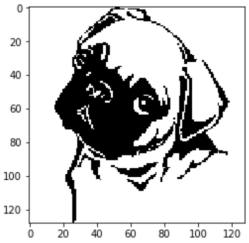
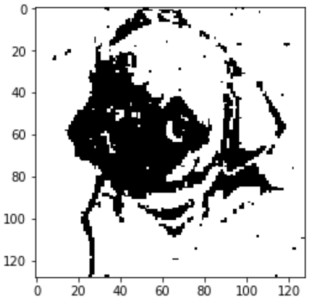


Figure 2: *This is the denoised image with the ICM approach (β=4.4, η=1). The noise level was 10% and the noise type was Gaussian noise. All three approaches resulted in similar visual images for Gaussian noise, and they all output identical results and therefore only this image is provided.*

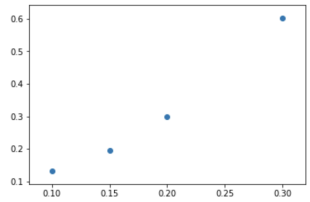
distribution is conditioned on every other dimension. An important difference between ICM and Gibbs sampling for our current model is that with the latter we don't blindly select a value for our latent variables simply on what has higher probability based on our model, but we value the ratio of the two probabilities, not just which one is higher. For example, if the probability of a pixel's latent representation to be 1 is 0.51 and its probability to be -1 is 0.49, ICM would certainly force it to be 1, whilst in the case of Gibbs sampling the probability of setting it as 1 or -1 would be almost equal. The result can be seen in Figure 3. The denoised image is much better than in ICM, even with a high noise proportion.

1.2.2 Question 3

Implementing Gibbs with random samples instead of sequential ones produces similar results to the ones in the previous question. This assumes that we still choose 128×128 random pixels in each iteration T. However, instead of selecting every pixel once for every iteration, as we did in the previous question, it is now likely that we will select some pixels more than once for some iterations or not at all for others. This means that the algorithm now needs more iterations to converge to a satisfyingly denoised image. We graphically compare the number of iterations in Question 7.



(a) *Denoised salt and pepper image with Gibbs (β=1, η=2).*



(b) *Mean Squared Error of denoised images with ascending percentage noise levels with Gibbs.*

Figure 3

1.2.3 Question 4

Gibbs sampling for our Ising model works similarly to the ICM implementation but there is always a probability that some pixels will be set to a value that is not preferred according to our prior and likelihood. This means that it's extremely unlikely that no pixels will change after an iteration. Therefore, it is a good idea to manually stop the code after a set number of iterations when the algorithm seems to have converged. This seems to happen after about 10 iterations.

1.3 Deterministic Inference

1.3.1 Question 5

We begin our discussion by providing an intuitive explanation of the KL divergence. Suppose we have two distributions, p(x) and q(x). By writing down the ratio p(x)/q(x) for some x we get a measurement of how likely x is to have come from one of the two distributions. If the ratio is larger than 1, then x is more likely to have come from the numerator, otherwise it is more likely to have come from the distribution in the denominator. We will now generalize our discussion for an arbitrary number of data points. Since the data points are independently and identically distributed, we can calculate the average ratio of a data point to have come from one of the two distributions by considering the ratio given all data points.

$$\frac{p(x_1, x_2, ...x_n)}{q(x_1, x_2, ...x_n)} = \prod_i^n \frac{p(x_i)}{q(x_i)} \tag{1}$$

To simplify the calculations we can take a log and write the product as a sum:

$$\sum_i^n \log \frac{p(x_i)}{q(x_i)} \tag{2}$$

Since we changed to logs, now if the ratio is larger than 0, the values are more likely to have come from the distribution in the numerator, otherwise, they are more likely to have come from the one in the denominator.

Let's now suppose that we start sampling values from q(x) instead of uniformly. We are now sure that the distribution that they are more likely to have come from is the one we sampled them from. Therefore, the above ratio will now quantify how likely the data are to have come from the other distribution, p(x). If we now consider all possible samples ie when $n \to \infty$, the sum is equivalent to taking the expectation over q(x) which can be written as follows:

$$\lim_{n \to \text{inf}} \frac{1}{n} \sum_{i=0}^{n} \log \frac{p(x_i)}{q(x_i)} =$$
$$\mathbb{E}_{x \sim p(x)} \{\log \frac{p(x_i)}{q(x_i)}\} =$$
$$\int p(x) \log \frac{p(x)}{q(x)} dx$$

This is the definition of KL divergence.

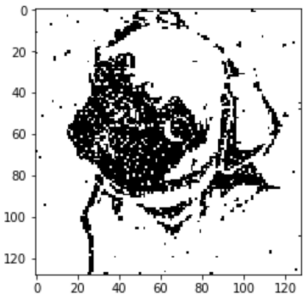
We can therefore infer that equation is not symmetric since when we are sampling from q(x) if q(x) places a lot of mass on values that p(x) places no mass on, then the data cannot be explained by p(x) at all, and the ratio in our divergence measure will be very large, since our denominator is close to 0. The fact that p(x) does not place mass where q(x) does, does not imply that q(x) does not place mass where p(x) does, which means that values that can be explained very well by p(x) might also be able to be explained by q(x) partially. This is why the KL divergence is not symmetric.

1.3.2 Question 6

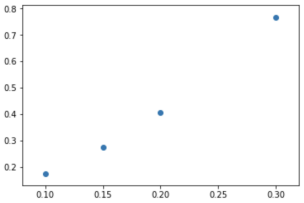
We now implement a deterministic algorithm to derive intractable posterior distributions rather than probabilistically sampling from them. The central idea of variational Bayes is to find a distribution $q(x)$ to approximate the intractable posterior $p(x \mid y)$. We quantify how well the posterior is approximated with $q(x)$ by taking their KL-Divergence $KL(q(x) \mid\mid p(x \mid y))$. In Variational Bayes, we attempt to minimize the KL divergence so that $q(x)$ is as closely approximated to $p(x \mid y)$ as possible. In the implementation, as the posterior distribution can only be evaluated at $x_i = 1$ or $x_i = -1$, we compute q as an $M \times N$ dimensional grid, such that its i^{th} element when flattened to a 1D array represents $q(x_i = 1)$. Since we want to make a decision over a binary random variable for each of the dimensions/pixels of the image, we infer the initial image as follows:

$$x[i,\ j] = \begin{cases} 1, \text{ if } q[i,\ j] > 0.5 \\ -1, \text{ otherwise} \end{cases}$$

In figure 4, we observe that Variational Bayes is succesful at inferring the original latent variables even though, comparing Figure 3b with Figure 4b we see that it is not as succesful as Gibbs. However, as we discuss further in Question 7, this is because of the nature of Variational Bayes, which makes it faster to converge but also not as accurate since we are sampling from an approximate posterior.



(a) *Denoised salt and pepper image with Variational Bayes (η=1.35).*



(b) *Mean Squared Error of denoised images with ascending percentage noise levels with Variational Bayes.*

Figure 4

1.3.3 Question 7

In the case of Gaussian noise, Gibbs sampling and Variational Bayes work equally well despite the level of noise added. ICM also seems to be successful. On the other hand, when salt and pepper noise is added, it is interesting to observe how low the mean squared error drops with each method and the number of iterations required. We observe that even though ICM converges quickly it has the highest error rate compared to the other models. This makes sense since it's the simplest approach and very strict in the way that it sets pixels as black and white. Gibbs and Random Gibbs are the best performing methods with the lowest error rate. This is to be expected since MCMC methods will eventually converge to the correct answer given enough iterations. As mentioned earlier, random Gibbs needs a few more iterations to converge than sequential Gibbs since it is likely that some pixels will not be updated in some iterations. We will now compare ICM and Gibbs with Variational Bayes. To begin with, the result given by Variational Bayes is very close to the original denoised image, but not an exact replication of it, since the distribution q(x) that we sample from is not precisely equal to the intractable posterior p(x). Instead, it's a different function, whose form is known, and its differences with our actual posterior are estimated by minimizing the KL divergence. Therefore, even if we let the algorithm run for a lot of iterations, since we are merely approximating the form of the posterior, we can't hope to obtain the exact latent variables. However, the advantages of variational methods, as we can observe in Figure 5, involve their speed. Variational Bayes tends to immediately converge to a denoised figure very close to the original one much quicker than the other methods.

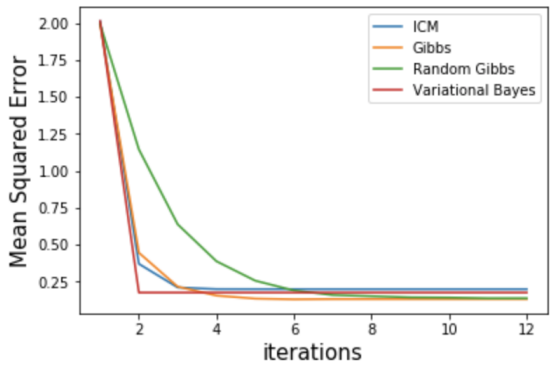


Figure 5: *Mean Squared Error of the different methods.*

(a) *Initial image used for image segmentation.*

(b) *Masked image used for image segmentation.*

Figure 6

1.4 Image Segmentations

1.4.1 Question 8

In this question, we describe the procedure followed to extract the foreground from an image. The approach followed involves some data preprocessing first, and then the actual separation of the background and the foreground occurs.

Data preprocessing To begin with, we will need to define the prior and the likelihood. The prior will stay as before, so that it captures the intuition that "If a pixel is surrounded by background (resp. foreground) pixels and given our belief that images are smooth, then the pixel we are examining will probably be a background (resp. foreground) pixel." To define the likelihood we need some more ground work. It needs to give a probability value for each pixel colour conditioned on "foreground" or "background". More formally, we need to define $p(pixel \mid fg)$ and $p(pixel \mid bg)$, where "pixel" is a three-element column vector whose individual values range from 0 to 255. This raises the question of how it is feasible to define such a likelihood. First, we require that the user defines a mask on top of a portion background pixels and another distinct mask on top of a portion of foreground pixels, as shown in Figure 6b. This allows to build two histograms over pixel colours, each defining a probability that a certain colour is background or foreground respectively. Such a histogram can be build by iterating through the pixels underlying a mask and increasing the counter in the respective bin. Defining such a histogram over the whole space of colours would require 255^3 bins. Even if it was feasible to allocate such a data structure and extract values efficiently from it, the distribution defined in it would be almost uniform, and thus not very helpful. Therefore, we make the relaxation of using only twenty bins. We define those bins by clustering the pixels under the two masks using K-Means, and assigning each pixel to the bin whose euclidean distance from it is the least. Both histograms are then normalized. We can now sample from either of the two histograms to compute the likelihood for some pixel being a background or foreground pixel.

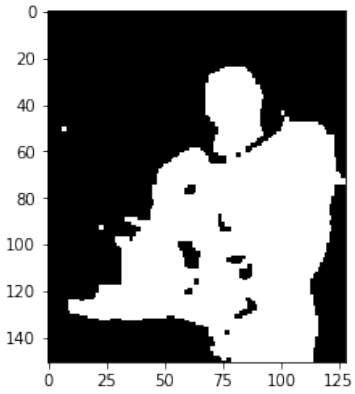


Figure 7: *The converged values of the array X are visualised, representing the separated foreground and background.*

Extracting the foreground We are now ready to separate the foreground and background pixels. We formulate this goal by assigning values -1 and +1 to the elements of the array x, each representing a pixel and -1, +1 representing whether the corresponding pixel is believed to belong to the background or foreground respectively. The array X is initialized to contain arbitrary values. We formulate the log of the energy function by adding the log prior and log likelihood for each pixel, and we iterate through the pixels. In each iteration of each pixel we sample from the posterior by comparing the probabilities:

$$\ln p(y_i \mid x_i \in fg) p(x_i \in fg, x_N(i)) \text{ and} \quad (3)$$

$$\ln p(y_i \mid x_i \in bg) p(x_i \in bg, x_N(i)) \quad (4)$$

where fg and bg are the sets containing the foreground and background pixels respectively. We thus update the value for x_i based on the above comparison. We iterate many times through each pixel until convergence is reached. Then, the array X is outputted as an image. It can be seen in Figure 7

2 Variational Auto-encoder

2.0.1 Question 9

We will first discuss how variational autoencoders work, and then we will compare with variational bayes methods. The variational autoencoder consists of two neural networks, an encoder and a decoder. The encoder takes inputs with a very large number of dimensions, in our case a black and white hand- written digit, and maps them to a lower dimensional latent space. Therefore, the encoder's role is to approximate the posterior of our model which attempts to successfully recover the latent variables x, given observations z. However, instead of outputting a discrete encoding, the gaussian MLP encoder that is used outputs distributions a mean and standard deviation. Therefore, even for the same input, the encoding will vary every time. This means that our latent space is now continuous.

The decoder does the opposite. Given the latent low dimensional representation produced by the encoder, it will produce a high dimensional representation of it, similar to the original. In our scenario, we are using a Bernoulli decoder which takes one of the latent variables as input and generates a high dimensional image, a black and white digit that's based on the proximity to the different means of the latent representations. Each pixel from the generated image will be a bernoulli distribution that affects if it will be black or white.

The loss function used to train the network includes the negative log likelihood which ensures that the image generated by the decoder given latent variables will be similar to the image that was encoded. The KL divergence is also included in the loss function, and ensures that the the outputs of the encoder will be distributions similar to a standard Gaussian distribution. This encourages the encodings to be closer to each other. The combination of the two parts of the loss function allow clusters with similar latent variables to be close to each other, but still grouped based on similarity.

In general, it is easy to observe that both variational autoencoders and variational bayes are approaching the same problem from different perspectives. The encoder that we try to optimize in the former case is the same as the approximative posterior in variational bayes while the decoder is the likelihood. Similarly, the KL divergence plays a different result in each method, as discussed in the previous part. Furthermore, in mean field variational bayes, we still need to compute the expectation with respect to the approximate posterior, which is also intractable sometimes. With variational autoencoders, we can efficiently approximate the marginal inference of the latent variables and therefore the approximative posterior using gradient ascent. More specifically, when we introduce the function $q(z|x)$ that is used to approximate the posterior in the case of variational autoencoders we do not have to assume that it necessarily factorizes over the latent variables z, parametrized by μ, which means that it is a more general algorithm that works in more situations.

2.0.2 Question 10

Variational autoencoders don't just learn how to recover the images that were encoded, but are also able to generate new ones that are similar to the observed ones. Due to the two components in its loss function, the model learns both how to successfully reproduce the inputs but also how to cluster the encodings in the latent space so that they are close to each other, by including the KL divergence where one of its components is a standard Gaussian distribution. In our case, this encourages different hand written versions of the number "6" to be close to each other, but also not very far away from written versions of the number "8". Furthermore, since the encodings in variational autoencoders are gaussian distributions themselves because of the form of the encoder, and because, as it was just explained, the latent space in general is encouraged to be a gaussian distribution itself, we can sample random points from the approximated posterior and generated new versions of hand written numbers. We can therefore observe versions of handwritten digits that are similar but not included to the ones in the dataset.

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[2] Cotra, Marko. *Making Sense of the Kullback–Leibler (KL) Divergence*. Medium.com, Medium, 12 Feb. 2017, medium.com/@cotra.marko/making-sense-of-the-kullback-leibler-kl-divergence-b0d57ee10e0a.

[3] D. P. Kingma and M. Welling. *Auto-encoding variational Bayes*. In Proceedings of the International Conference on Learning Representations, 2014.

[4] Geron Aurelien. *Hands-on Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2017.