Inference

Eddie Jones 36147 and Kheeran Naidu 35838

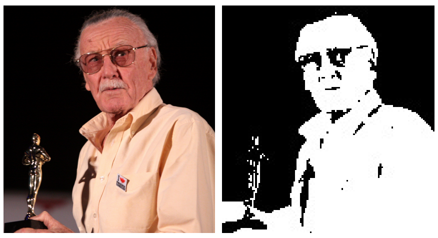

Figure 1: *Stan Lee*

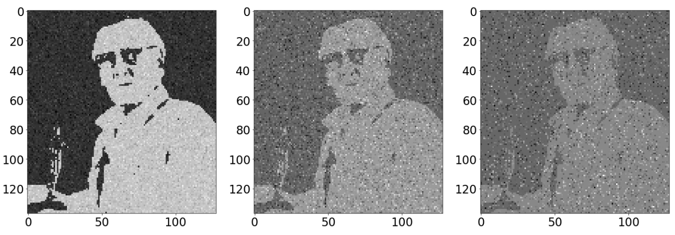

Figure 2: *Three different noise levels*

1 Approximate Inference

1.1 Data

The image we have chosen to work with is the first image in figure 1. After applying a grey-scale filter, re-sizing and adjusting the levels we ended up with the second image. We also map the pixel values from 0-255 to -1-1, this makes noise easier as first we have a real value oppose to an integer and the mean can be 0.

This image should be illustrative as it consists of large continuous regions which matches our prior assumption about pixel neighbourhoods as well as some more complex regions.

1.2 Iterative Conditional Modes

Question 1 The iterative conditional modes method (ICM) works by first initialising the latent variable $\mathbf{x} = \mathbf{x}_0$. For each dimension x_i of the vector we consider whether it would be more or less likely to be a $+1$ or -1 under the joint distribution $p(\mathbf{x}, \mathbf{y})$ we then assign it accordingly and proceed to the next dimension $(i+1)$. This whole process is iterated many times to increase the chance of arriving at the most likely vector, effectively we are considering an axis parallel or rectilinear gradient decent.

The joint distribution is computationally intractable, however, even without the normalisation factors. We can get around this by assuming a form of $L_i(x_i)$ as $\eta x_i y_i$, as well as $w_{ij} = \beta$, we can then using the properties of exponentials to get:

$$\frac{1}{Z_1} \prod_{i=1}^{N} e^{L_i(x_i)} \frac{1}{Z_0} e^{\sum_{j \in \mathcal{N}(i)} w_{ij} x_i x_j} = \tag{1}$$

$$e^{\left(\beta \sum_{\{i,j\}} x_i x_j + \eta \sum_{i=1}^{N} x_i y_i\right)} \tag{2}$$

[Bis06]

Where $\{i, j\}$ are neighbouring indices. As we only need to compare probabilities for ICM we can reduce this further by ignoring the exponential. First we tested our model with Gaussian noise with three different levels shown in figure 2 The first with 70% of pixels altered with $\sigma^2 = 0.1$, the second 50% with $\sigma^2 = 0.5$, and the last 30% with $\sigma^2 = 1$.

We can extract the original image from the first noise using the parameters $\mathbf{x}_0 = \mathbf{0}, \beta = 0, \eta = 1$ and after only 1 iteration (For the remaining test we shall use the same initial vector unless specified otherwise). The second set of noises we can get very close (98.9% of pixels correct) to the original using $\beta = 0.2, \eta = 0.8, T = 4$. For the final image we ran it with $\beta = 0.25, \eta = 0.5$ for 8 iterations and produced the 3rd image in figure 3, which although visually distorted it has 97.8% of pixels the same as the original. When dealing with binary values visual similarity doesn't necessarily correspond with numeric similarity, making the cleaning-up process more complex to measure. It also shows that with certain types of noise we need drastically different numbers of iterations to find the original, and that the parameters necessary are not intuitive.

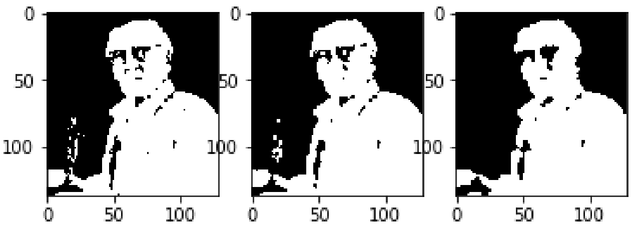

Figure 3: *Respective images in figure 2 denoised with ICM.*

With the salt 'n' pepper noise the algorithm wasn't as successful as seen in figure 4. This we generated with $\beta = 0.2$, $\eta = 1.0$ and 10 iterations. Here it has had some success in the foreground and background where the image is more consistent but only produced an accuracy of 77.6% across the entire image. (The salt 'n' pepper noise is generated with $p = 0.3$, values much higher become impossible to denoise effectively)

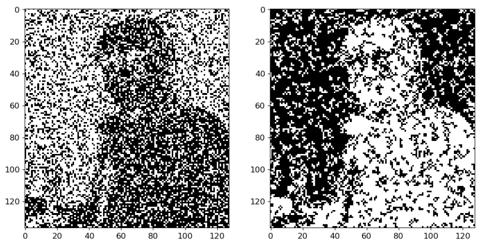

Figure 4: *Salt 'n' pepper noise with ICM denoising on the right*

1.3 Stochastic Inference

Question 2 The Gibbs sampling Ising model allows us to sample from $p\left(\mathbf{x}|\mathbf{y}\right)$ and in the process compute an approximation for the previously intractable formula for the maximum likelihood:

$$\arg\max_{\mathbf{x}} p\left(\mathbf{x}|\mathbf{y}\right).$$

This is done by considering, for each index i, the conditional probability of $x_i = 1$, and then using an uniform random distribution to decide whether to keep this value or to instead set $x_i = -1$. Although we are using a stochastic element, the relative frequency of sampling will still correspond to the actual probability. Unlike with the ICM model, with Gibbs sampling we are comparing the probabilities not with each other, but to a normalised uniform distribution so it isn't apparent that we can simply ignore any normalising factors. Although, we can make the computation easier by re-factorising the probabilities, and in this form we can also see the normalising factors cancel.

$$p\left(x_i|\mathbf{x}_{\neg i},\mathbf{y}\right) = \tag{3}$$

$$\frac{p(y_i|x_i=1)p\left(x_i=1,x_{\mathcal{N}(i)}\right)}{p(y_i|x_i=1)p\left(x_i=1,x_{\mathcal{N}(i)}\right)+p(y_i|x_i=-1)p\left(x_i=-1,x_{\mathcal{N}(i)}\right)} \tag{4}$$

For the specific case of our model we can calculate the previous equation using:

$$p\left(x_i,x_{\mathcal{N}(i)}\right) = \frac{1}{Z_0}e^{\beta\sum_{j\in\mathcal{N}(i)}x_ix_j} \tag{5}$$

$$p\left(y_i|x_i\right) = \frac{1}{Z_1}e^{\eta x_i y_i} \tag{6}$$

When applied to the same noisy images as before we can produce accurate cleaned-up images using $\beta = 0.2, \eta = 1, T = 5$ for the first image to correct 98.4% of pixels, the second with $\beta = 0.2, \eta = 0.8, T = 10$ to 96.4% and the final image with $\beta = 0.25, \eta = 0.8, T = 15$ to 95.3%,as seen in figure 5.

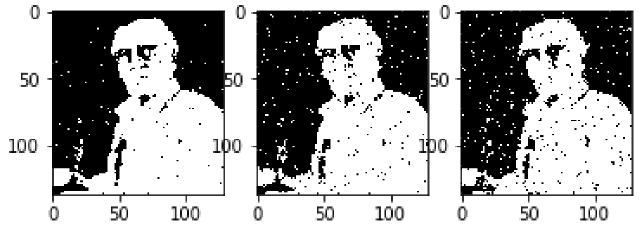

Figure 5: *Respective images in figure 2 denoised with Gibbs.*

We also tested our Gibbs sampling method with the salt 'n' pepper noise as seen in figure 6, with the same parameters as ICM the model produced an accuracy of 80.3%, which is slightly better than the ICM model unlike with the Gaussian noise, this makes sense as the more stochastic element of Gibbs gets less caught up with individual pixels being wrong.

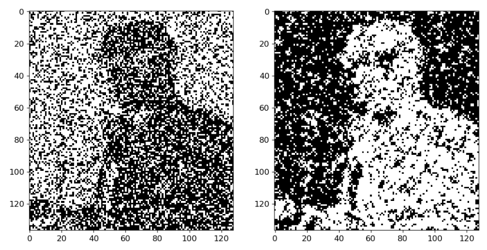

Figure 6: *Salt 'n' pepper noise with Gibbs denoising on the right*

As we can see from these results the variance of the noise is more of factor than number of pixels altered in the models ability to reconstruct the images. This intuitively makes sense as if the pixels are only minutely changed, even if most of them are, the likelihood function still makes it easy to correct this. With more varied noise I higher number of iterations is required as well as a higher value of β, this is because the likelihood function is less useful at determining the original pixel and so we must rely more heavily on our assumption about the image space, that neighbouring pixels are similar. This also means it is less accurate in less consistent areas of the image, because of this dependency on β.

The accuracy of the images produced is generally lower than with ICM. The ICM model is deterministic and is guaranteed to arrive at some local maximum (in some unbounded number of iterations), however it is extremely slow to run as for each iteration we have to compute both the probability that $x_i = 1$ and $x_i = -1$ joint with all $\mathbf{x}_i$ and $\mathbf{y}$ which is computationally expensive. Gibbs sampling however only requires one of these values **conditional** on the others, and therefore is much faster to compute. On the other hand, the random nature of its behaviour means it won't always give the same result in the same number of iterations. It also has the advantage that by chance it may arrive at a global maximum irrespective of initial conditions.

Question 3 In the Gibbs sampling method we cycle though the indices of the vector $\mathbf{x}$ in order. If on the other hand we pick a index to change at each iteration at random we can get very similar results but it requires more (effective) iterations. In the original method for each T we actually perform M×N iterations and so for the random index sampling we shall also. For example with the Gaussian noise using the same parameters and inputs as before we get 97.3% accuracy for the first image 96.0% accuracy for the second and 95.3% for the last, as seen in figure 7. This makes sense in some respect as in the original method we are guaranteed to modify and potentially update each pixel T times, here however, no such guarantee exists so we may end up modifying one particular pixel a disproportionate number of times and make very little progress on the overall image. On the other hand it may be the case that a different parameter set means this method is actually faster.

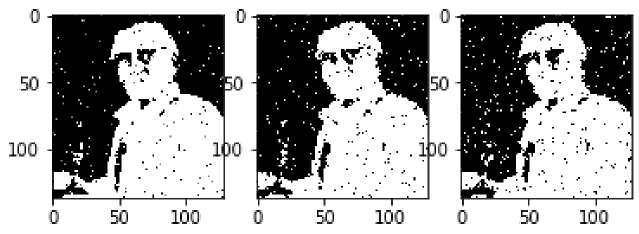

Figure 7: *Respective images in figure 2 denoised with Random Gibbs.*

Question 4 It is not the case that increasing the number of iterations necessarily increases the accuracy. Although initially we would expect to see this trend as pixels are repeatedly updated based on the observed data and our assumption about clustering within the image, depending on the relative weight of parameters, it is possible that this clustering effect grows and grows and distorts the image more as more iterations are undergone.

1.4 Deterministic Inference

Question 5

$$\text{KL}(p(\mathbf{x})||q(\mathbf{x})) = \int p(\mathbf{x})log\frac{p(\mathbf{x})}{q(\mathbf{x})}d\mathbf{x} \tag{7}$$

We can see by analysing the equation of KL-divergence (eq 7) that it is an asymmetric measure. It measures the relative average information gain of distribution p(**x**) relative to q(**x**). Intuitively we can see how there will always be a positive relative average information gain; for KL(p(**x**)||q(**x**)), when p(**x**) is smaller than q(**x**), the log of the ratio will be negative, however, the probabilistic weight of the ratio would be small as-well, and conversely in the case when p(**x**) is larger, the log of the ratio will be positive with a large weight (figure 8 & 9). Formally, we use Jensen's inequality to show that $\text{KL}(p(\mathbf{x})||q(\mathbf{x})) \geq 0$ [Mac03]. In the task of fitting a distribution q(**x**) to p(**x**), we are effectively minimising KL(p||q); finding a dist q(**x**) that gives p(**x**) as close to a zero relative information gain $\forall$**x**, which is only possible when p(**x**) = q(**x**) since the log of the ratio (=1) will always be zero (i.e. figure 9 when x=2,6). In that case $\text{KL}((p(\mathbf{x})||q(\mathbf{x})) = \text{KL}((q(\mathbf{x})||p(\mathbf{x}))$.

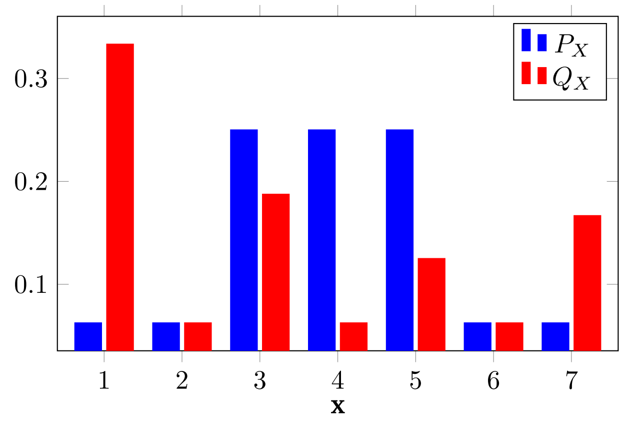

Figure 8: *Probability density for finite* $P_X = \{\frac{1}{16}, \frac{1}{16}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{16}, \frac{1}{16}\}$ and $Q_X = \{\frac{2}{6}, \frac{1}{16}, \frac{3}{16}, \frac{1}{16}, \frac{1}{8}, \frac{1}{16}, \frac{1}{6}\}$

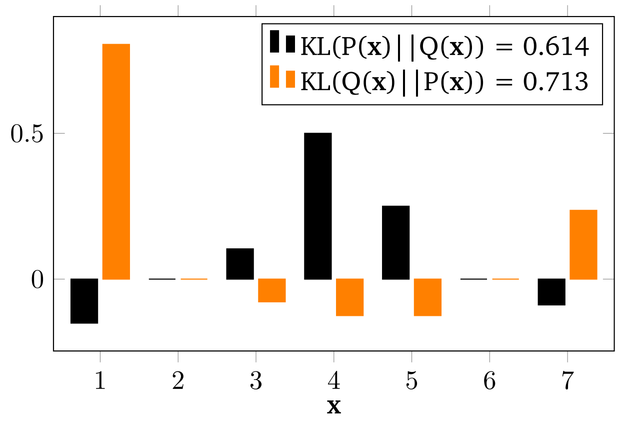

Figure 9: *Corresponding relative information gain*

Question 6

$$logp(\mathbf{y}) \geq -\text{KL}(q(\mathbf{x})||p(\mathbf{x}|\mathbf{y})) + logp(\mathbf{y}) \tag{8}$$

Variational Bayes works by trying to fit some distribution q(**x**) to the posterior p(**x**|**y**). Given some data **y**, we try to calculate the log evidence, log p(**y**), using Jensen's inequality. Knowing that a logarithm is a concave function, we derive eq 8 and see that log p(**y**) is tight bounded when $\text{KL}(q(\mathbf{x})||p(\mathbf{x}|\mathbf{y})$ is minimised, implying that we want the distributions q(**x**) and p(**x**|**y**) to be most similar.

To do this, we try find a q(**x**) that minimises $\text{KL}(q(\mathbf{x})||p(\mathbf{x}|\mathbf{y}))$, however, the posterior, p(**x**|**y**) is intractable, so we derive the eq 27 in the question paper [Ek18] and maximise the ELBO. We do this by using Mean Field Approximation to find an optimum distribution q(**x**) which minimises the KL divergence and successfully denoises the 3 noisy images from figure 2 into the 3 images in figure 10. The 3 images were denoised using the same parameters giving 99.5% correct for the first image, 98.5% correct for the second and 96.1% for the third, as seen in 10

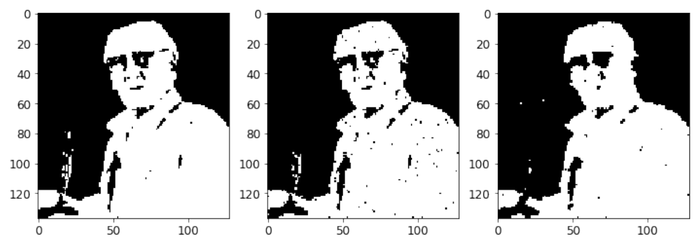

Figure 10: *Respective images in figure 2 denoised with Variational Bayes.*

Question 7

$$p(\mathbf{x}|\mathbf{y}) = \frac{e^{\left(\beta\sum_{\{i,j\}}x_ix_j+\eta\sum_{i=1}^{N}x_iy_i\right)}}{p(\mathbf{y})} \tag{9}$$

All 3 methods are different approaches to dealing with a posterior with an intractable evidence and so rely on the same posterior equation. This means that the prior is the same and that the assumption encoded in each method is equivalent. By studying the posterior (eq 9), we see that the assumptions that we have encoded are the contributions of the neighbourhood of a pixel and its corresponding observed value. We assign β[1] and η respectively to determine how much we actually care about the contributions. We can also go a step further in the analysis and say that the importance in determining our posterior is the ratio $\beta : \eta$ as seen in figure 11 using Variational Bayes to denoise the most noisy of the 3 images in figure 2.

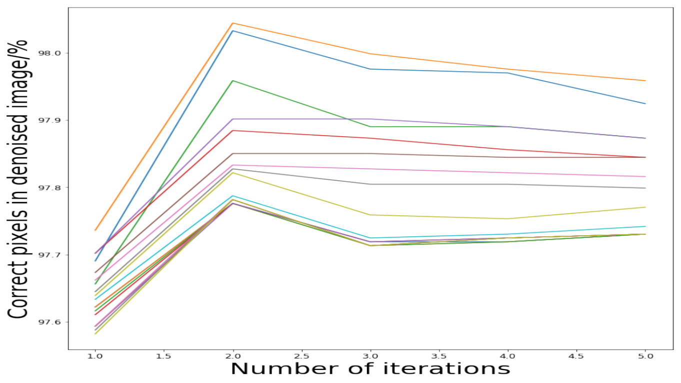

Figure 11: *Different values of beta and eta with ratio 3:5. The y-axis range is 97.6-98.0% which shows the close similarity.*

In all 3 methods, we can improve the convergence of the algorithms by accessing the pixels in a random order, without repeats, rather than sequential accessing. This, however, decreases the speed of convergence and increases the amount of computation per iteration due to randomly[2] accessing the array of pixels on each iteration. In addition, we can also initialise the latent variable to the observed variable in order to increase the likelihood of finding a global optimum and increase the rate of convergence.

Undoubtedly, there are also differences in the methods, mostly between ICM/Gibbs and Variational Bayes. In the ICM method, we calculate the probability that our latent variable x_i is either 1 or -1 and assign x_i the value with higher probability. In the Gibbs Sampling method, we additionally sample a random variable to give us a better representation of the probability when selecting the value of x_i, rather than using a binary cutoff method. Gibbs basically builds upon ICM. In the Variational Bayes method, however, we approach denoising the image by finding an approximate distribution to the posterior (using Mean Field Approximation). The main difference is that ICM and Gibbs are stochastic approaches and Variational Bayes is a deterministic one. A stochastic approach is good as theoretically they give an exact answer given the right set up and enough iterations, however it is computationally slower.

A deterministic approach is good because it will generally approach a local minimum faster, however it allocates too much weight to exploitation and not enough to exploration, that is depending on the initial setup it is unlikely to find the global minimum as with gradient descent. In comparison a stochastic approach always has the capacity to find the optimum solution because of its ability to explore the space fully, although it does in general take longer to do so due to 'non-optimum' local improvements at each iteration.

[1]This encodes our assumption that the image is equally likely to have horizontal, vertical and diagonal changes in the image gradient.

[2]Perfect randomness is impossible to sample from.

Prop	σ^2	Denoise	β	η	T	correctness/%
0.7	0.1	ICM	0.2	1	5	99.6
		Gibbs				98.4
		Var Bayes				99.5
0.5	0.5	ICM	0.2	0.8	10	98.9
		Gibbs				96.4
		Var Bayes				98.5
0.3	1	ICM	0.25	0.8	15	97.8
		Gibbs				95.3
		Var Bayes				96.1

Table 1: *Comparison of the different denoising methods.*

By studying table 1 we learn the efficacy of each method. We see that Gibbs sampling is the least efficient of the images in our test set. ICM and Variational Bayes are comparable in terms of efficacy. We see in figure 12 the affect of denoising the left image (noisy with salt and pepper at prop=0.3) into the middle one, using ICM and the right one using Variational Bayes, both with $\beta = 0.2$ and $\eta = 1$. This shows us that Variational Bayes is much better with images that have more sporadic noise.

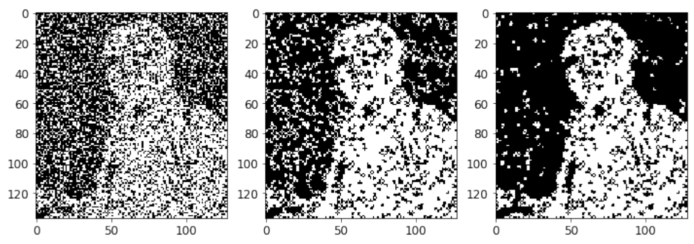

Figure 12: *Comparison of ICM (middle) to VB (right) using salt and pepper noise (left).*

1.5 Image Segmentation

Question 8

The problem of image segmentation can be approached using the same models, although the latent variable x will be interpreted with a different semantic, in this case $x_i = 1$ indicated that the ith pixel as coming from the foreground and -1 the background. The assumption about neighbouring pixels still hold and all we need to do to update the model is change the likelihood. The new likelihood function will determine whether a given y_i value is likely to come from the foreground or background by comparing it to a histogram of 'foreground' and 'background' pixels (which are determined by the mask, x, from the previous iteration) we then update our mask accordingly. Our first attempt uses the histogram to store a pseudo-brightness of the pixel, the norm of its RGB value scaled to the range $[0, 1]$.

We chose to implement this algorithm using Gibbs sampling. A particular issue we encountered was a good initially mask, x_0. This is a problem as, unlike with previous tasks, using a constant mask to begin with (either representing that all pixels are in the fore or background respectively) produces likelihood functions where all pixels are fore or background as the alternative histogram is empty. Using histograms to build likelihood function also introduces a new parameter, the number of bins, a too high quantity of bins will stratify the pixels too finely and exclude relevant pixels and a too small quantity won't be able to differentiate different parts of the image correctly.

Using a random initially mask (generated from a binomial with probability 0.9), 10 bins and a β value of 0.05 we produced a foreground-background distinction as seen in figure 14 after 10 iterations. This method has made some progress but clearly hasn't totally distinguished Stan Lee and his award from the background. It is also the case that the initial mask has a big impact on the result, especially with a limited number of iterations. The sample outputs bellow are the best results found with these parameters. This is a typical frequentist view where we haven't imposed any prior assumption on the space of foreground-background pixels and so an initial mask of all background will classify any point to the background too.

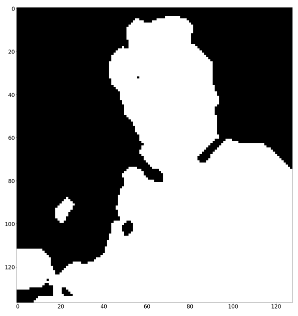

Figure 13: *The foreground-background mask from brightness histogram*

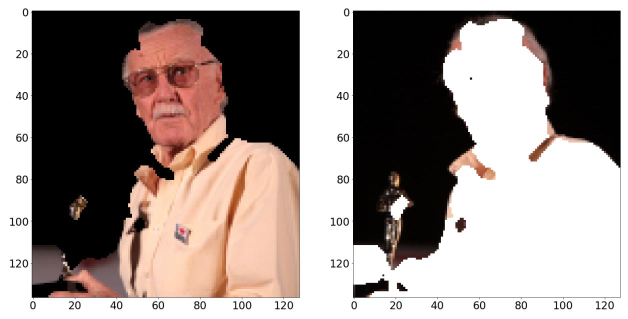

Figure 14: *The foreground and background extracted using the mask, reffig:mask1*

An alternative approach might be to use k-nearest neighbours or a Gaussian mixture model to classify points, this has the advantage of considering all dimensions oppose to projecting the space down and potentially losing information. To update these models on each iteration is powerful but computationally costly so instead we choose to use a k-means clustering to create "histogram" bins out of which centroid is closest, this way the clustering maintains some of the structure of the image but is still fast to run.

This produced more accurate results much faster as seen in figure 15 and 16 which were generated with 10 bins, $\beta = 0.01$, and only 5 iterations. Although it is still dependant on the inital mask.

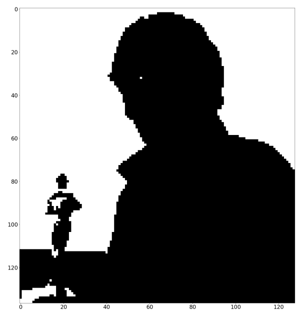

Figure 15: *The foreground-background mask from k-means histogram*

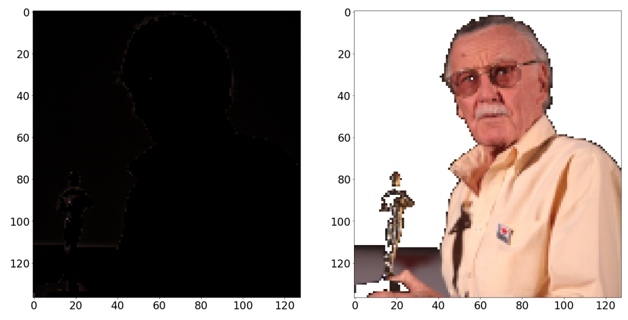

Figure 16: *The foreground and background extracted using the mask, 15*

2 Variational Auto-encoder

Question 9 [Jor18]
An auto-encoder uses a neural network to compress a very high dimension input space into a lower one, with a huge variety of applications. This is a form of dimensionality reduction or feature selection/extraction but more of a black-box approaches than traditional methods. The use of a neural network enables a more complex higher-level extraction process through use of layers. There are two parts, an encoding set of layers which reduces the input down to the latent variable set (the bottleneck) and a decoding set of layers which attempts to reconstruct it. We can now measure the success of our neural network by considering some reconstruction loss metric between the input and output layers, from this the neural network can learn in a similar approach to back-propagation. We can apply this technique to other tasks as well beyond compression, for example image segmentation by changing how we measure the success of the reconstruction.

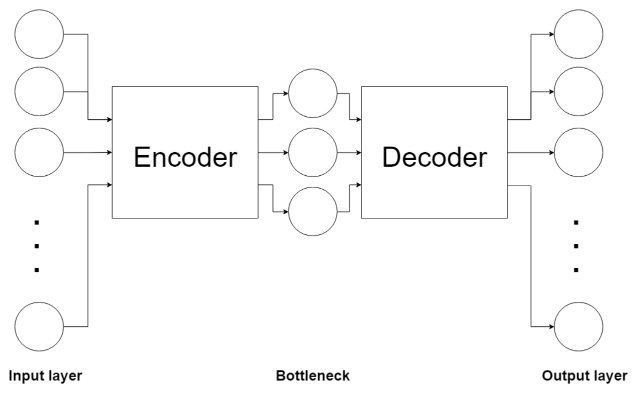

Figure 17: *An example auto-encoder*

A variational auto-encoder is very similar except instead of a normal bottleneck layer being a fixed vector it remodels it using a distribution (specifically a multivariate normal). This is done by introducing two layers into our bottleneck: the first containing the parameters of the distribution (a mean vector and co-variance matrix in the case of a multivariate normal distribution), the second pseudo-layer doesn't depend on the weights of the first as with a standard neural network but instead samples from this distribution before sending the result into the decoder. [Jor18]

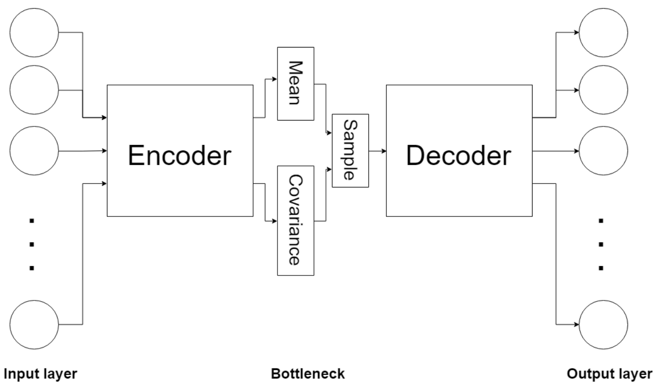

Figure 18: *An example variational auto-encoder*

To train a variational auto-encoder we now need a new loss function composed of two terms in a similar way to variational Bayes. The first term encodes for reconstruction loss (similar to a traditional auto-encoder however here we use the expectation of the output) represented through the log probability of our samples, $\mathbf{x}$, conditional on our latent variables, $\mathbf{z}$. The second term encodes the KL divergence to prevent the distribution exploding.

$$\mathbb{E}\left[\log p\left(\mathbf{x}|\mathbf{z}\right)\right] - \mathrm{KL}\left(q\left(\mathbf{z}|\mathbf{x}\right) \parallel p\left(\mathbf{z}\right)\right) \tag{10}$$

Even with the adjusted loss function we cannot run back propagation through the network because we are sampling in the bottleneck. To this end we parameterise the sample $\mathbf{z}$ as $\boldsymbol{\mu} + \boldsymbol{\Sigma}\boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{N}\left(\mathbf{0}, \mathbf{I}\right)$ and the other parameters are fixed. Now we can run back propagation from our decoder's output through to $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ by not including the stochastic element, $\boldsymbol{\epsilon}$, in the back propagation chain.

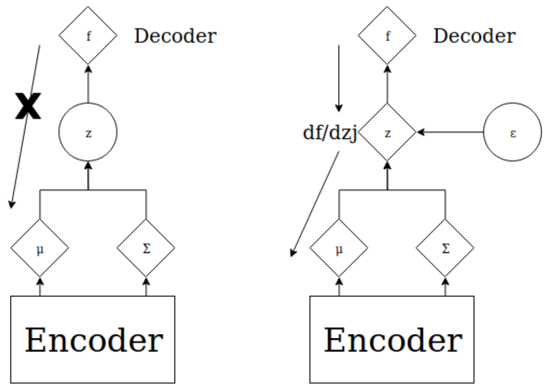

Figure 19: *Reparameterisation dependencies. Diamond: Deterministic node, Circle: Random node*

We don't make any assumption about the encoding and decoding function beyond the number of dimensions. Within the bottleneck the stochastic element represents the assumption that there is some normally distributed noise in the input but we don't assume any of the parameters. In the reparameterisation step to abstract the stochastic element out we have to make the assumption that each dimension is non-correlated which implies independence in the case of a Gaussian.

Question 10

The MNIST experiment trains a generative model for the digits. It samples a value from a normal distribution, with mean μ and covariance Σ, to use as the parameter for decoding into an image of a digit.

The auto-encoder is divided into the encode and decode sections. In the case of the given MNIST experiment, both sections have equal number of layers and there are 2 hidden layers and 1 output layer per section. The encoder takes in an image (and so the input layer will have a node for each pixel), it then goes through two hidden layers each of 500 nodes. The first layer has an ELU (Exponential Linear Unit) activation function, which behaves similarly to the ReLU where it is linearly for positive values except unlike ReLU it is exponential for negative values. This has the advantage of learning fast but can produce negative outputs. The second layer uses a $\tanh$ activation function, this has the advantage of strong gradients resulting again in faster learning near optimum but it also comes with the vanishing gradient problem, that is the derivatives can be so small learning effectively halts[17]. However in this case we use the dropout function to prevent from happening. Finally the encoder outputs a 2-dimensional mean and a 2x2 covariance matrix which are used to sample the output $\mathbf{z}$. This network is refereed to as a Gaussian Multilayer Perceptron (MLP) as it outputs the parameters of a Gaussian distribution.

The decoder again has 2 hidden layers, as with the encoder these each are of size 500 nodes. However the order of activation functions is reversed; first $\tanh$ and then ELU (both with the dropout function). The output layer has the same dimension as our image but is first passed through a sigmoid function to ensure the pixel values are in the right domain, a binary image. The decoder is a Bernoulli MLP as the output, each pixel value, can be seen as a Bernoulli probability distribution with the reconstructed image composed of p values which represents the probability of it being a white. Applying a sigmoid gives the current binary image representation of our sampled $\mathbf{z}$ decoded in the current neural network.

$$y_t \log y + (1 - y_t)\log(1 - y_t) \tag{11}$$

We use this image to get the marginal likelihood, which is the mean of the cross entropy (eq 11) in a Bernoulli distribution, and evaluating our loss function [3], loss = -elbo. We then update the weights of our neural network accordingly in order to minimise the loss; this maximises the elbo which is what we need in order to find a distribution that minimises KL-divergence. We repeat this training process (depicted in figure 20 with our entire train data set to find the μ and σ

[3]To prevent calculating log 0 which is undefined, we min cut values less than $\epsilon = 1 \times 10^{-6}$ to ϵ and max cut values greater than $1 - \epsilon$ to $1 - \epsilon$

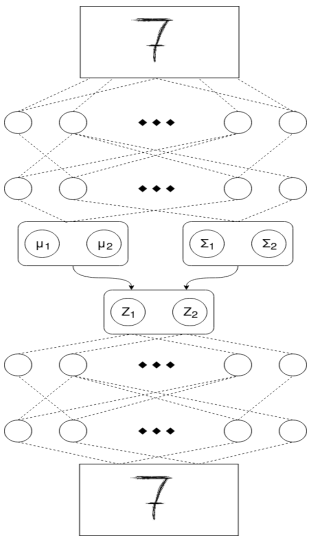

Figure 20: *Visualisation of the training of MNIST [Y L98] with the ideal output.*

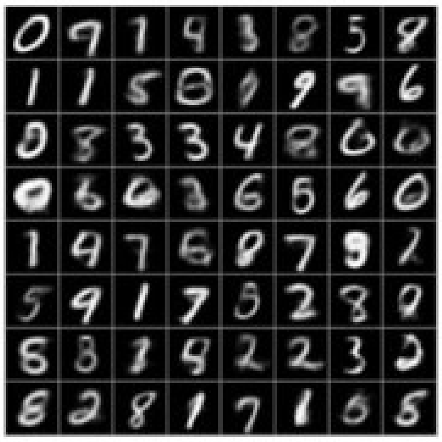

Figure 21: *Samples from the trained VAE with different samples of* $\mathbf{z}$ *[Moh17].*

which best represents our encoder, which we have assumed to be Gaussian. Our trained model, is now a generative model for handwritten digits. In order to generate a digit, we just sample $\mathbf{z}$ from the Gaussian and propagate it through the Bernoulli decoder to get a binary image of a handwritten digit with examples shown in figure 21.

Bibliography

[17] *Activation Functions*. 2017. URL: `https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html`.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. ISBN: 0387310738.

[Ek18] Carl Henrik Ek. *Inference - Coursework question paper*. 2018. URL: `https://github.com/carlhenrikek/COMS30007/blob/master/Labs/Inference/inference.pdf`.

[Jor18] J Jordan. *Introduction to Autoencoders*. 2018. URL: `https://www.jeremyjordan.me/autoencoders/`.

[Mac03] D Mackay. *Information Theory, Inference, and Learning Algorithms, CUP*. 2003. URL: `http://www.inference.phy.cam.ac.uk/itprnn/book.html`.

[Moh17] Felix Mohr. *Teaching a Variational Autoencoder (VAE) to draw MNIST characters*. 2017. URL: `https://towardsdatascience.com/teaching-a-variational-autoencoder-vae-to-draw-mnist-characters-978675c95776`.

[Y L98] C Burges Y LeCun C Cortes. *MNIST handwritten digit database*. 1998. URL: `http://yann.lecun.com/exdb/mnist/`.