Inference

Assignment in COMS30007 Machine Learning

Carl Henrik Ek

November 4, 2017

Welcome to the second assignment in the machine learning course. You will present the assignment with by a written report that you should submit on SAFE, try to leave some space for submission and don't wait till to close to the deadline. From the report it should be clear what you have done and you need to support your claims with results. You are supposed to write down the answers to the specific questions detailed for each task. This report should clearly show how you have drawn the conclusions and come up with the derivations. Your assumptions, if any, should be stated clearly. For the practical part of the task you should not show any of your code but rather only show the results of your experiments using images and graphs together with your analysis. You should still submit your code though, but you are free to use whatever language that you want.

Being able to communicate your results and conclusions is a key aspect of any scientific practitioner. It is up to you as a author to make sure that the report clearly shows what you have done and what your understanding is. Based on this, and only this, we will decide if you pass the task. No detective work should be needed on our side. Therefore, neat and tidy reports please! Each report can be up to 5 pages long.

I very much recommend you to get used to LaTeX to write your report. It is an amazing tool that you will find very useful in your further endeavors as a scientist. You can download the style-file from `here`. You are **not** allowed to alter the margins nor the font-size but you are free to use `\vspace{}` as much as you like. If you decided to use another formatting engine try to mimic the style file provided

If anyone is an `emacs` conouseur I recommend using the excellent `org-mode` and the `ox-latex` export engine to prepare your report.
The grading of the assignments will be as follows,

50% Question 1-4

70% Question 1-7

80% Question 1-8

90% Question 1-10

This means that this is your potential top mark if all the questions are answered correctly and the discussion is as it should be. For the 90% mark, you have to do the amortised learning bit but as you have been provided with the code it is your intuitions that are important. The last 10% I am using as golden sprinkle that I hand out to things that I think are especially well done so there is absolutly the possibility to get a 100% mark.

Abstract

In the first assignment we looked at how we can create models that allows us to parametrise and factorise a distribution over the observed data domain. We saw that we can make decisions from the posterior distribution of the model which we reached by combining the model with observed data. For many different models though it is not feasible to compute the posterior in closed form most commonly because the marginal likelihood or the evidence is not analytically or computationally tractable. So how do we proceed? Well one possibility is to look for a point estimate rather than the full distribution and proceed with a Maximum Likelihood or a Maximum-a-Posteriori estimate. However, this should really be our last resort as these methods will at best tell us what it believes the "best" approach is but will not at all quantify what "best" means. Further our assumptions in this case does not reach the data which means they are at best regularisers. This means that for such an inference scheme we cannot make a choice if we should trust the model or not, nor can we make a choice on how well the model actually describes the data. The more sensible approach is to try and approximate the intractable integrals and here there are two main approaches, either stochastic or deterministic methods. They both have their benefits and negatives. A stochastic approach is simple to formulate but importantly it is hard to assess how well we are doing and in many ways it is considered one of the "black arts" of machine learning. Deterministic approaches are usually very efficient but they will never be exact. In this assignment we will look at both of these approaches but to keep it safe we will avoid most of the voodoo of sampling and focus on deterministic methods.

1 Approximate Inference

The task of inference in a machine learning model is the task of combining our assumptions with the observed data. In specific we have a set of observed data $\mathbf{Y}$ which have been parametrised by a variable $\boldsymbol{\theta}$ the task requires us to use Bayes rule to reach the posterior $p(\boldsymbol{\theta}|\mathbf{Y})$,

$$p(\boldsymbol{\theta}|\mathbf{Y}) = \frac{p(\mathbf{Y}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathbf{Y})}.$$

The challenging part of the relationship above is the marginal likelihood or the evidence, which is the probability of the observed data when *all* assumptions have been propagated through and integrated out,

$$p(\mathbf{Y}) = \int p(\mathbf{Y}, \boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta}.$$

In the first assignment we looked at situations where we can avoid calculating the marginal likelihood by exploiting conjugacy, however, for certain cases it is simply not possible as this integral is intractable, either computationally but quite often it is analytically intractable. In order to proceed we have to make sacrifices and approximate this integral. But in order for the assignment to get underway we need to have a model to play around with that will exemplify the different approaches.

In this part we are going to look at the rather useful task of image restoration, in specific we are going to work with binary images which have been corrupted by noise and we are supposed to clean them up. The task is exactly the same if you want to perform image segmentation rather than denoising and I will explain the latter as a task for extra marks.

1.1 The Model

Images are one of the most interesting and easily availible sources of data, images contain a lot of information and they can be aquired in an unitrusive manner with very cheap sensors. Our task here is to build a model of images, in specific of binary or black-and-white images. Images are normally represented as a grid of pixels y_i however the images we observe are noisy and rather will be a realisation of an underlying latent pixel representation x_i. Now to make our computations a bit easier, lets say that white is encoded by $x_i = 1$ and black with $x_i = -1$ and that the greyscale values that we observed $y_i \in (0, 1)$. We will write our likelihood

on this form,

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z_1}\prod_{i=1}^{N} e^{L_i(x_i)}, \tag{1}$$

where $L_i(x_i)$ is a function which generates a large value if x_i is likely to have generated y_i and Z_1 is a factor that ensures that $p(\mathbf{y}|\mathbf{x})$ is a distribution. We have further assumed that the pixels in the image are conditionally independent given the latent variables $\mathbf{x}$.

The next part is to think what a sensible prior would be, what do we actually know about images? One important aspect of images that makes them, well images is that there is a significant correlation between neighbouring pixels. What do we know about this relationship? Well, lets say that we see one white pixel, what do we believe the most likely colour of the pixel to the right to be? If I had to guess I would probably say white as I think that images have more contious segments of one colour compared to switches between colours. So this is now prior information, an assumption that we want to quantify in terms of a probability. We can write down this as follows,

$$p(\mathbf{x}) = \frac{1}{Z_0}e^{E_0(\mathbf{x})}, \tag{2}$$

where again $E_0(\mathbf{x})$ is a function that is large the configuration of $\mathbf{x}$ is something that we believe is likely and small otherwise and Z_0 a normalising term to ensure that $p(\mathbf{x})$ is a distribution. If we follow our previous reasoning and say that a pixel depends on its neighbouring pixels only we can write $E_0(\mathbf{x})$ as function of the following form,

$$E_0(\mathbf{x}) = \sum_{i=1}^{N}\sum_{j\in\mathcal{N}(i)} w_{ij}x_i x_j, \tag{3}$$

where $\mathcal{N}(i)$ specifies the set of nodes that are neighbours to node i. Remember that $x_i \in [-1, 1]$ this means that $x_i x_j$ will be 1 if the nodes have the same label and -1 if the nodes have different labels. The scalars w_{ij} are our parameters that we can control the strength of our prior with, where a large value w_{ij} implies that node $x_i x_j$ are nodes that we really believe should have the same label. Now we have our final model and can describe the joint distribution,

$$p(\mathbf{x}, \mathbf{y}) = p(\mathbf{y}|\mathbf{x})p(\mathbf{x}) = \frac{1}{Z_1}\prod_{i=1}^{N} e^{L_i(x_i)}\frac{1}{Z_0}e^{\sum_{j\in\mathcal{N}(i)} w_{ij}x_i x_j}. \tag{4}$$

We can also write up the graphical model for the model which is shown in Figure 1. The model that we just have described is referred to as a Markov Random Field with a Ising prior. This model was initially described in physics to study nearby magnets where the latent variable was their "direction". However, it turns out that they are very good models for images in many tasks.

1.2 Inference

The task we will study in this paper is to given a noisy observation $\mathbf{y}$ recover the latent variables $\mathbf{x}$ that have generated the observations. This means that we want to reach the posterior distribution $p(\mathbf{x}|\mathbf{y})$ to do so we have to compute Baye's rule,

$$p(\mathbf{x}|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{x})p(\mathbf{x})}{p(\mathbf{y})}.$$

The denominator could be computed as follows,

$$p(\mathbf{y}) = \sum_{\mathbf{x}} p(\mathbf{y}|\mathbf{x})p(\mathbf{x}).$$

For any type of sensible size of image this summation is not computationally tractable. What we want to sum over is all possible values that $\mathbf{x}$ can actually take, i.e. we want to test **all** possible binary images. If

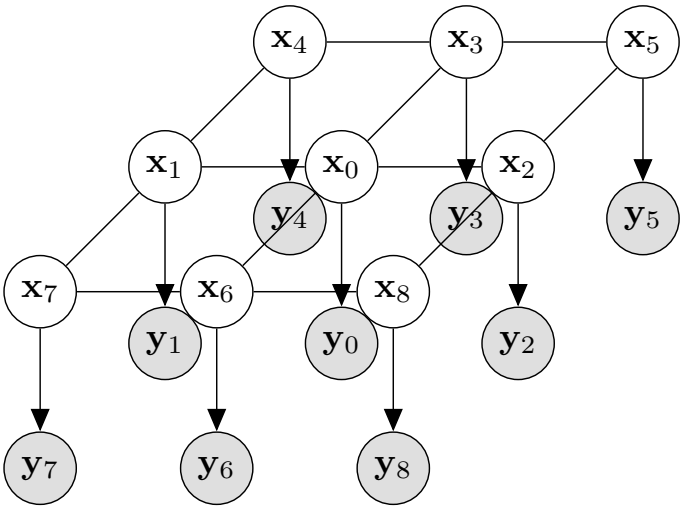

Figure 1: *Above is the graphical model of the MRF we will use for the images. Note that we have connected the latent variables with lines and not arrows, that is because we do not specify these as conditional probabilities but rather joint probabilities.*

we have an image of size 10 it consists of 100 different pixels. The number of combinations that they can take is therefore 2^{100}. This is the number of terms in the marginalisation above and its a *big* number. This means that it is simply computationally intractable to compute Baye's rule for any sensibly sized image, say something with 3-4 megapixels, and we need to perform some form of approximation to preoceed. As a side-note think back on the previous assignment where we could use conjugacy to avoid these calculations, we summed over a space with *infinite* number of terms to reach the posterior when we did Gaussian processes without actaully directly computing the evidence. Now think how nice conjugacy is.

We will now proceed to look at three different approaches to inferring the true pixel values in the above the first method is just a simple cordinate-wise gradient approach while the two others are more principled approximations. The assignment should be fairly straight forward to code up but what I want you to try and do, where I think you will learn the most, is by playing around with the parameters, initialisations etc. so that you get an intuitive understanding for what is going on. Right lets get started!

1.3 Data

First we need some data to work with, you can use any image that you want as a starting point, if in doubt I can definetly recommend images of pugs, the work great. To make our life a bit easier we use greyscale images rather than colour. You can use the Imagemagic to convert between colour and greyscale and resize the image to something sensible.

Code

```
convert -resize 128x <image-in> <image-out>
convert <image-in> -set colorspace Gray -auto-level -threshold 50% <image-out>
```

Once the image is converted we can load it into python and create a noisy version of it. The code below has two different types of noise, either gaussian noise or 'salt-and-peppar' noise which flips the pixel values.

Code

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.misc import imread

def add_gaussian_noise(im,prop,varSigma):
    N = int(np.round(np.prod(im.shape)*prop))

    index = np.unravel_index(np.random.permutation(np.prod(im.shape))[1:N],im.shape)
    e = varSigma*np.random.randn(np.prod(im.shape)).reshape(im.shape)
    im2 = np.copy(im)
    im2[index] += e[index]

    return im2
def add_saltnpeppar_noise(im,prop):
    N = int(np.round(np.prod(im.shape)*prop))
    index = np.unravel_index(np.random.permutation(np.prod(im.shape))[1:N],im.shape)
    im2 = np.copy(im)
    im2[index] = 1-im2[index]

    return im2

# proportion of pixels to alter
prop = 0.7
varSigma = 0.1

im = imread('text.png')
im = im/255
fig = plt.figure()
ax = fig.add_subplot(131)
ax.imshow(im,cmap='gray')

im2 = add_gaussian_noise(im,prop,varSigma)
ax2 = fig.add_subplot(132)
ax2.imshow(im2,cmap='gray')
im2 = add_saltnpeppar_noise(im,prop)
ax3 = fig.add_subplot(133)
ax3.imshow(im2,cmap='gray')
```

In order to process the images we are also likely to need some helper code to access the image. In specific our prior requires us to compute the neighbours of a specific node, the code below computes the *4-neighbourhood* of a node so it does not include the diagonals. You might try to extend the code below and include the diagonals as well as this should improve the results.

Code

```
def neighbours(i,j,M,N,size=4):
      if size==4:
            if (i==0 and j==0):
                  n=[(0,1), (1,0)]
            elif i==0 and j==N-1:
                  n=[(0,N-2), (1,N-1)]
            elif i==M-1 and j==0:
                  n=[(M-1,1), (M-2,0)]
            elif i==M-1 and j==N-1:
                  n=[(M-1,N-2), (M-2,N-1)]
            elif i==0:
                  n=[(0,j-1), (0,j+1), (1,j)]
            elif i==M-1:
                  n=[(M-1,j-1), (M-1,j+1), (M-2,j)]
            elif j==0:
                  n=[(i-1,0), (i+1,0), (i,1)]
            elif j==N-1:
                  n=[(i-1,N-1), (i+1,N-1), (i,N-2)]
            else:
                  n=[(i-1,j), (i+1,j), (i,j-1), (i,j+1)]

            return n
      if size==8:
            print('Not yet implemented\n')

            return -1
```

Now we have most of the useful code we need and its time to move on the the specific inference algorithms.

1.4 Iterative Conditional Modes (ICM)

The first approach we should take is something called Iterative Conditional Modes you can read about it in Section 8.3.3 in [1]. This approach works like this, we will first initialise the latent variables to something, then we will fix all variables except for one and see what is the most likely state for this one to be in given that we assume all others to be correct. We will then iteratively do this for all the nodes and if we manage to go through one pass of all nodes without changing then we have reached a local minima. However, just to get some control of things, I'm going to run it a fixed set of iterations in the code below, but I do recommend that you keep a flag for changes so that you can bail early, or know if you have found a local minima. You can see the algorithm in Algorithm 1 where $\mathbf{x}_{\neg i}$ implies all $\mathbf{x}$ except x_i.

Algorithm 1 Iterative Conditional Modes for Ising Model

procedure IMAGE DENOISING WITH ICM
  $\mathbf{x} \leftarrow$ initialisation
  for $\tau = 1 \dots T$ **do**
    for $i = 1 \dots N$ **do**
      if $p(x_i = 1, \mathbf{x}_{\neg i}, \mathbf{y}) > p(x_i = -1, \mathbf{x}_{\neg i}, \mathbf{y})$ **then**
        $x_i = 1$
      else
        $x_i = 1$
  return $\mathbf{x}$

Question 1

Implement ICM for a binary image that you have corrupted with noise, show the result for a set of images with different noise levels. How many passes through the nodes do you need until you are starting to get descent results?

1.5 Stochastic Inference

Now we should pick up a bit more advanced algorithm to try and find the latent variables from data. What we will do here is to implement a simple Gibbs sampler Ch 11.3 [1]. Gibbs sampling is often quite easy to implement so it is one of the first approaches you often try to get something out of a model and see if it is worth developing a specific taylored inference scheme.

1.5.1 Basic Sampling

The idea behind sampling is that we want to compute and expectations over a function where the closed form is intractable,

$$\mathbb{E}_{p(\mathbf{z})}[f] = \int f(\mathbf{z})p(\mathbf{z})\mathrm{d}\mathbf{z}.$$

We now try to convert the integral to a discrete sum of values that are draw from $p(\mathbf{z})$ as,

$$\hat{f} = \frac{1}{L}\sum_{l=1}^{L} f(\mathbf{z}^{(l)}).$$

The important thing to note here is that the approximation will depend on us drawing "good" samples, this is really what sampling is about different strategies to get informative samples.

1.5.2 Markov Chain Monte Carlo

One of the strategies to get more efficient sampling is referred to as Markov Chain Monte Carlo methods or MCMC for short. You will see them pop up in many different topics, if you are taking computer graphics in TB2 you will bump into them as high fidelity graphics is a lot about intractable integrals. MCMC was developed as a part of the Manhattan project in the development of the first nuclear bomb where nummerical solutions where sought to complicated or intractable problems. The idea behind MCMC is to let the sequence of samples come from a Markov chain such that when we draw a new sample we take the previous evaluations into consideration. Note that this does not mean that the samples are not independent according to the distribution we want to sample from. In specific we will use a proposal distribution $q(\mathbf{z}|\mathbf{z}^{(\tau)})$ to draw samples from where $\mathbf{z}^{(\tau)}$ is the current state of our sampling chain.

1.5.3 Gibbs Sampling

Gibbs sampling is probably the most straight-forward type of MCMC method. It is widely used and very easy to implement. The idea behind a Gibbs sampler is if we have a distribution $p(\mathbf{z})$ that we wish to draw samples from we will draw samples from each dimension in turn where we condition on the other dimensions. In specific we will sample from,

$$p(z_i|\mathbf{z}_{\neg i}),$$

where $\mathbf{z}_{\neg i}$ is all dimensions of $\mathbf{z}$ except for i. We will then replace z_i with our samples and "rotate/cycle" through the variables. The idea behind a Gibbs sampler is outlined in Algorithm 2. Now lets try and relate this to our specific task that of image denoising.

Algorithm 2 Gibbs Sampling

procedure GIBBS SAMPLER FOR $p(\mathbf{z})$
  $\mathbf{z} \leftarrow$ initialisation
  for $\tau = 1 \dots T$ **do**
    $z_1^{(\tau+1)} \approx p(z_1 | z_2^{(\tau)}, \dots, z_N^{(\tau)})$
    $z_2^{(\tau+1)} \approx p(z_2 | z_1^{(\tau)+1}, z_3^{(\tau)}, \dots, z_N^{(\tau)})$
    $\vdots$
    $z_N^{(\tau+1)} \approx p(z_2 | z_1^{(\tau)+1}, z_2^{(\tau+1)}, \dots, z_{N-1}^{(\tau+1)})$
  return z

1.5.4 Gibbs Sampling in an Ising Model

We have now described how to sample from a general multivariate distribution $p(\mathbf{z})$ now we want to try and use this scheme to our MRF Ising model. In specific we are interested in sampling from the unreachable posterior $p(\mathbf{x}|\mathbf{y})$. The key thing underpinning Gibbs sampling is that even though the multivariate posterior might be unreachable, it should be much simpler to get the posterior over a single variable. If this is possible then we can run Gibbs sampling by rotating through the posterior over a single variable x_i. To begin lets formulate this distribution over a general node x_i,

$$p(x_i|\mathbf{x}_{\neg i}, \mathbf{y}) = \frac{p(\mathbf{x}, \mathbf{y})}{p(\mathbf{x}_{\neg i}, \mathbf{y})}. \tag{5}$$

To proceed we need to compute the marginal likelihood above, this turns out to be rather easy as we are working with binary data,

$$p(\mathbf{x}_{\neg i}, \mathbf{y}) = \int p(\mathbf{x}, \mathbf{y}) \mathrm{d}x_i = \sum_{x_i \in [1,-1]} p(x_i, \mathbf{x}_{\neg i}, \mathbf{y}) \tag{6}$$

$$= p(x_i = 1, \mathbf{x}_{\neg i}, \mathbf{y}) + p(x_i = -1, \mathbf{x}_{\neg i}, \mathbf{y}). \tag{7}$$

So now lets say that we want to compute the posterior over $x_i = 1$ we can write this up as,

$$p(x_i = 1|\mathbf{x}_{\neg i}, \mathbf{y}) = \frac{p(x_i = 1, \mathbf{x}_{\neg i}, \mathbf{y})}{p(x_i = 1, \mathbf{x}_{\neg i}, \mathbf{y}) + p(x_i = -1, \mathbf{x}_{\neg i}, \mathbf{y})}, \tag{8}$$

which we can evaluate as we know $\mathbf{y}$ and $\mathbf{x}$. However, what we want to do is to sample from the posterior over x_i but as we do not even know what form the distribution is we are going to do a simple trick. We will evaluate $p(x_i = 1|\mathbf{x}_{\neg i}, \mathbf{y})$ and then we will draw a random number z from the uniformly from $(0, 1)$ and then we will pick x_i from this distribution where we use the posteriors as proportions. Below is a couple of example that hopefully explains things clearly,

$p(x_i = 1\|\mathbf{x}_{\neg i}, \mathbf{y})$	z	x_i^{sample}
0.5	0.7	-1
0.5	0.2	1
0.1	0.05	1
0.1	0.2	-1

You could now go straight on to implement the Gibbs sampler but it would most likely be very slow as you would have to evaluate distributions with very many parameters every iterations. To speed things up

Algorithm 3 Gibbs Sampling Ising Model

procedure GIBBS SAMPLER FOR $p(\mathbf{x}|\mathbf{y})$
  $\mathbf{x}^{(0)} \leftarrow$ initialisation
  for $\tau = 1 \dots T$ **do**
    for $i = 1 \dots N$ **do**
      $p_i = p(x_i = 1|\mathbf{x}_{\neg i}^{(\tau)}, \mathbf{y})$
      $t \sim \text{Uniform}(0, 1)$
      if $p_i > t$ **then**
        $x_i^{\tau+1} = 1$
      else
        $x_i^{\tau+1} = -1$
  return $\mathbf{x}^{(T)}$

we can use the structure that exists in the problem. Lets write up our posterior,

$$p(x_i = 1|\mathbf{x}_{\neg i}, \mathbf{y}) = \frac{p(x_i, \mathbf{x}_{\neg_i}, \mathbf{y})}{p(\mathbf{x}_{\neg i}, \mathbf{y})} \tag{9}$$

$$= \frac{p(y_i|x_i = 1) \prod_{j \neq i} p(x_j|y_j) p(x_i = 1, \mathbf{x}_{\neg i})}{p(y_i|x_i = 1) \prod_{j \neq i} p(x_j|y_j) p(x_i = 1, \mathbf{x}_{\neg i}) + p(y_i|x_i = -1) \prod_{j \neq i} p(x_j|y_j) p(x_i = -1, \mathbf{x}_{\neg i})} \tag{10}$$

$$= \frac{p(y_i|x_i = 1) p(x_i = 1, \mathbf{x}_{\neg i})}{p(y_i|x_i = 1) p(x_i = 1, \mathbf{x}_{\neg i}) + p(y_i|x_i = -1) p(x_i = -1, \mathbf{x}_{\neg i})} \tag{11}$$

$$= \frac{p(y_i|x_i = 1) p(x_i = 1, \mathbf{x}_{\mathcal{N}(i)})}{p(y_i|x_i = 1) p(x_i = 1, \mathbf{x}_{\mathcal{N}(i)}) + p(y_i|x_i = -1) p(x_i = -1, \mathbf{x}_{\mathcal{N}(i)})} \tag{12}$$

where $\mathbf{x}_{\mathcal{N}(i)}$ is the set of nodes that are in the neighbourhood of x_i. This is a much smaller computation where we have exploited the structure in the problem in two ways, first that the likelihood factorises, this means we only have to compute one single term and secondly that the prior also factorises into the Markov blanket of x_i. Now we are ready to implement the Gibbs sampler for our Ising model. The code that you need to write should follow the Algorithm 3.

Question 2

Implement the Gibbs sampler for the image denoising task. Generate images with different amount of noise and see where it fails and where it works. My result is shown in Figure 2.

Question 3

There is nothing saying that you should cycle through the nodes in the graph index by index, you can pick any different order and you do not have to visit each node equally many times either. Alter your sampler so that it picks and updates a random node each iteration. Are the results different? Do you need more or less iterations? To get reproduceable results fix the random seed in your code with `np.random.seed(42)`.

Question 4

What effect does the number of iterations we run the sampler have on the results? Try to run it for different times, does the result always get better?

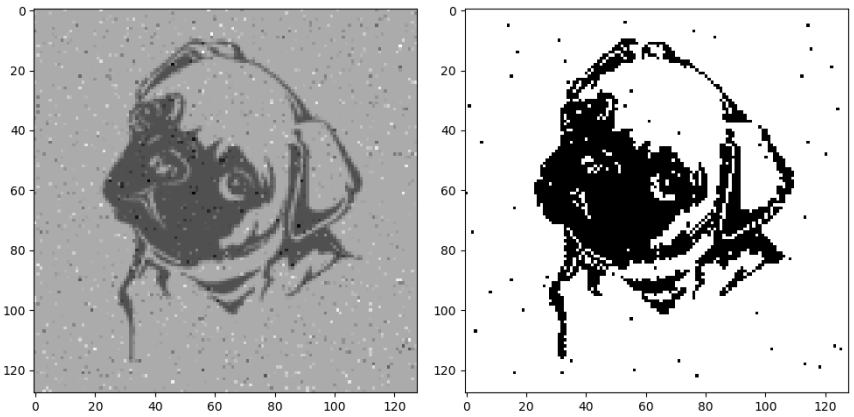

Figure 2: *The result of running the Gibbs sampling approach in the Ising model. The left image is the noisy version while the rightmost is the cleaned up version*

1.6 Deterministic Inference

Now we will move on to a deterministic approximation of the Ising Model. What we will do is to specify a surrogate model and try to fit this model so that it is as close as possible to the actual model. We will first go through the idea of Variational Bayes and then proceed to go through and look at the specific approach we will use for the Ising model.

Math

This derivation is long, I go through the full proof of, variational Bayes, mean-field and mean-field variational Bayes for the Ising model. I tried to be as complete as possible so everything should be self-contained. However, there is likely to be blunders in there so if you find something strange point it out to me. Importantly, you do not need to know any of this, its here if you are really interested in machine learning but none of the questions in the assignment is related to this. If you want to skip the derivations read 1.6.1 and then move ahead to 1.6.4. However, if I am allowed to say it myself this is quite beautiful so if you enjoy math I suggest you continue reading.

1.6.1 Variational Bayes

Inference is the task of fitting our model to some observed data, what we often do is to try to choose the model that maximises the evidence. If we have been given some data $\mathbf{y}$ and have some parameters θ to fit we wish to pick them such that,

$$\hat{\theta} = \text{argmax}_\theta p(\mathbf{y}).$$

As we know the evidence is often intractable to compute but lets see what we can do,

$$\log p(\mathbf{Y}) = \log \int p(\mathbf{Y}, \mathbf{X}) \mathrm{d}\mathbf{X} = \log \int p(\mathbf{X}|\mathbf{Y}) p(\mathbf{Y}) \mathrm{d}\mathbf{X} \tag{13}$$

$$= \log \int \frac{q(\mathbf{X})}{q(\mathbf{X})} p(\mathbf{X}|\mathbf{Y}) p(\mathbf{Y}) \mathrm{d}\mathbf{X}. \tag{14}$$

The strange thing here is the second row where we have added a distribution $q(\mathbf{X})$ to the equation. This is just a general distribution and because we add it in this form it will not change the integral at all. What we will do now is try to formulate a bound on this integral, in specific we will use something called the Jensen inequality. The Jensen inequality states that a line between two points on the curve will always be above

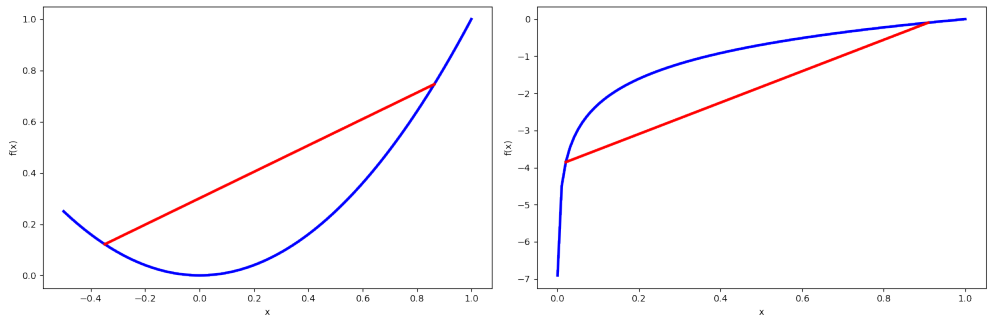

Figure 3: *The plot on the left shows a convex function in blue and line connecting two of the points on the function. The Jensen inequality implies that if you "move" a point along the red line it will always be above the blue. On the right the blue function is a logarithm and we can see that the reverse behaviour is true, the red line will always be below the blue. This means that we can say that, "the red line will always be a lower bound on the blue".*

the curve see Figure 3.

$$\begin{aligned} \lambda f(x_0) + (1-\lambda) f(x_1) &\geq f(\lambda x_0 + (1-\lambda) x_1) \\ x &\in [x_{min}, x_{max}] \\ \lambda &\in [0, 1]] \end{aligned}$$

Even though it might seem trivial it is a very useful property for dealing with probabilities. When we are marginalising variables from our model we are computing expectations, if we are computing an expectation of a convex function, the expectation of the function will always be an upper bound on the function applied to the expectations,

$$\mathbb{E}[f(x)] \geq f(\mathbb{E}[x]) \tag{15}$$

$$\int f(x) p(x) \mathrm{d}x \geq f\left(\int x p(x) \mathrm{d}x\right) \tag{16}$$

Where this is specifically important is when the function is a logarithm. As a logarithm is a concave function the inequality just flips around as can be seen in Figure **??**. This means that the logarithm of an integral is a upper bound on the log of the integral,

$$\int \log(x) p(x) \mathrm{d}x \leq \log\left(\int x p(x) \mathrm{d}x\right). \tag{17}$$

We will now exploit this result to try and find a bound on the intractable evidence,

$$\log p(\mathbf{Y}) = \log \int \frac{q(\mathbf{X})}{q(\mathbf{X})} p(\mathbf{X}|\mathbf{Y}) p(\mathbf{Y}) \mathrm{d}\mathbf{X} = \tag{18}$$

$$\geq \int q(\mathbf{X}) \log \frac{p(\mathbf{X}|\mathbf{Y}) p(\mathbf{Y})}{q(\mathbf{X})} \mathrm{d}\mathbf{X} \tag{19}$$

$$= \int q(\mathbf{X}) \log \frac{p(\mathbf{X}|\mathbf{Y})}{q(\mathbf{X})} \mathrm{d}\mathbf{X} + \int q(\mathbf{X}) \mathrm{d}\mathbf{X} \log p(\mathbf{Y}) \tag{20}$$

$$= -\mathrm{KL}\left(q(\mathbf{X}) || p(\mathbf{X}|\mathbf{Y})\right) + \log p(\mathbf{Y}). \tag{21}$$

The important part of the computation above is where we move the logarithm inside the integral by exploiting the bound. Importantly The first term after having split the integral into two is what is known as the Kullback-Leibler divergence Ch 1.6.1 [1]. This is a measure of "simiarity" between probability distributions.

It is not a metric as it, for example is not symmetric, but importantly it is only 0 if the two distributions are the same and positive in all other cases. This leads us to an important observation, if $q(\mathbf{X} = p(\mathbf{X}|\mathbf{Y})$ then the bound is tight. Importantly in order to reach the posterior distribution $p(\mathbf{X}|\mathbf{Y})$ we would have to compute the evidence. So this leads us to the central intuition of Variational Bayes, if we can pick a distribution $q(\mathbf{X})$ such that it is as close as possible to the posterior $p(\mathbf{X}|\mathbf{Y})$ we will have a good surrogate model, if it is exact, it is the same model. Therefore lets try to minimise the KL-divergence between the two distributions.

$$\begin{aligned}
\mathrm{KL}(q(\mathbf{X})||p(\mathbf{X}|\mathbf{Y})) &= \int q(\mathbf{X}) log \frac{q(\mathbf{X})}{p(\mathbf{X}|\mathbf{Y})} \mathrm{d}\mathbf{X} && (22)\\
&= \int q(\mathbf{X}) log \frac{q(\mathbf{X})}{p(\mathbf{X}, \mathbf{Y})} \mathrm{d}\mathbf{X} + log\ p(\mathbf{Y}) && (23)\\
&= H(q(\mathbf{X})) - \mathbb{E}_{q(\mathbf{X})}\left[log\ p(\mathbf{X}, \mathbf{Y})\right] + log\ p(\mathbf{Y}). && (24)
\end{aligned}$$

What we have done above is to write up the divergence as an expectation over the joint distribution and a term that only dependes on $q(\mathbf{X})$ and the evidence. If we now move the evidence over on the other side of the expression we will get this formulation,

$$\begin{aligned}
log\ p(\mathbf{Y}) &= \mathrm{KL}(q(\mathbf{X})||p(\mathbf{X}|\mathbf{Y})) + \underbrace{\mathbb{E}_{q(\mathbf{X})}\left[log\ p(\mathbf{X}, \mathbf{Y})\right] - H(q(\mathbf{X}))}_{\text{ELBO}} && (25)\\
&\geq \mathbb{E}_{q(\mathbf{X})}\left[log\ p(\mathbf{X}, \mathbf{Y})\right] - H(q(\mathbf{X})) = \mathcal{L}(q(\mathbf{X})). && (26)
\end{aligned}$$

As we know the KL-divergence has to be positive the remaining term is a lower-bound on the evidence. This is why this is referred to as the *ELBO-Evidence Lower BOund*.

So that was a whole lot of math but what does it all mean, what does this lead us to, has this actually solved anything? Well if we look at the formula above, what we want to find is $q(\mathbf{X})$ if we do find this, we know that it is an approximation of the true posterior $p(\mathbf{X}|\mathbf{Y})$ this is really useful, further the bound is specified by the computation of an expectation over the *joint* distribution of the data. 1) if we cannot formulate the joint distribution we are in bigger trouble and 2) we are allowed to choose the distribution $q(\mathbf{X})$ that we have to take the expectation over. Clearly this should be simpler to do.

Question 5

The KL-divergence is not a symmetric measure as,

$$KL(q(\mathbf{x})||p(\mathbf{x})) \neq KL(p(\mathbf{x})||q(\mathbf{x})).$$

If we look at the definition,

$$KL(q(\mathbf{x})||p(\mathbf{x})) = \int q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x})},$$

with the task of finding a distribution $q(\mathbf{x})$ that fits $p(\mathbf{x})$ as well as possible what will be the difference between $KL(q(\mathbf{x})||p(\mathbf{x}))$ and $KL(p(\mathbf{x})||q(\mathbf{x}))$. Think of where $q(\mathbf{x})$ would place probability mass in the different scenarios.
Hint: think of the fact that a number divided by something close to zero will be a very big number.

In the next part we will derive a specific family of approximations often referred to as mean-field approximations. They are often not particularly exact but they do work in most cases.

1.6.2 Mean Field Approximation

The mean-field approximation assumes that the approximative posterior factorises over all the variables as,

$$q(\mathbf{X}) = \prod_i q_i(\mathbf{x}_i).$$

We will proceed by deriving the bound related to this type of approximative distribution.

$$\mathcal{L}(q) = \int q(\mathbf{X}) \log \frac{p(\mathbf{Y}, \mathbf{X})}{q(\mathbf{X})} \mathrm{d}\mathbf{X} = \int \prod_i q_i(\mathbf{x}_i) \log \frac{p(\mathbf{Y}, \mathbf{X})}{\prod_k q_k(\mathbf{x}_k)} \mathrm{d}\mathbf{X} \tag{27}$$

$$= \int \prod_i q_i(\mathbf{x}_i) \left(\log p(\mathbf{Y}, \mathbf{X}) - \sum_k \log q_k(\mathbf{x}_k) \right) \tag{28}$$

For many types of models we want to update several distributions. What we will do now is to derive a scheme where we update one component in turn. Therefore we would like to re-write $\mathcal{L}(q)$ in such a manner that we can single out a single component,

$$\mathcal{L}(q_j) = \mathcal{L}(q_j) + \mathcal{L}(q_{\neg j}),$$

where $\neg j$ means all the components except for j.

$$\mathcal{L}(q_j) = \int \prod_i q_i(\mathbf{x}_i) \left(\log p(\mathbf{Y}, \mathbf{X}) - \sum_k \log q_k(\mathbf{x}_k) \right) \tag{29}$$

$$= \int_j \int_{\neg j} q_j(\mathbf{x}_j) \prod_{i \neq j} q_i(\mathbf{x}_i) \left(\log\, p(\mathbf{X}, \mathbf{Y}) - \sum_k \log\, q_k(\mathbf{x}_k) \right) \mathrm{d}\mathbf{x}_{\neg j} \mathrm{d}\mathbf{x}_j \tag{30}$$

$$= \int_j q_j(\mathbf{x}_j) \underbrace{\int_{\neg j} \prod_{i \neq j} q(\mathbf{x}_i) \log\, p(\mathbf{Y}, \mathbf{X}) \mathrm{d}\mathbf{x}_{\neg j}}_{\log\, f_j(\mathbf{x}_j)} \mathrm{d}\mathbf{x}_j$$

$$- \int_j q_j(\mathbf{x}_j) \int_{\neg j} \prod_{i \neq j} q(\mathbf{x}_i) \left(\log\, q_j(\mathbf{x}_j) + \sum_{k \neq j} \log\, q_k(\mathbf{x}_k) \right) \mathrm{d}\mathbf{x}_{\neg j} \mathrm{d}\mathbf{x}_j \tag{31}$$

$$= \int_j q_j(\mathbf{x}_j) \log\, f_j(\mathbf{x}_j) \mathrm{d}\mathbf{x}_j$$

$$- \int_j q_j(\mathbf{x}_j) \left(\log\, q_j(\mathbf{x}_j) \underbrace{\int_{\neg j} \prod_{i \neq j} q_i(\mathbf{x}_i) \mathrm{d}\mathbf{x}_{\neg j}}_{=1} + \underbrace{\int_{\neg j} \prod_{i \neq j} q_i(\mathbf{x}_i) \sum_{k \neq j} \log\, q_k(\mathbf{x}_k) \mathrm{d}\mathbf{x}_{\neg j}}_{\text{constant w.r.t. } q_j} \right) \tag{32}$$

$$= \int_j q_j(\mathbf{x}_j) \log\, f_j(\mathbf{x}_j) \mathrm{d}\mathbf{x}_j - \int_j q_j(\mathbf{x}_j) \log\, q_j(\mathbf{x}_j) \mathrm{d}\mathbf{x}_j + \text{const.} \cdot \underbrace{\int_j q_j(\mathbf{x}_j) \mathrm{d}\mathbf{x}_j}_{=1} \tag{33}$$

$$= \int_j q_j(\mathbf{x}_j) \log\, \frac{f_j(\mathbf{x}_j)}{q_j(\mathbf{x}_j)} \mathrm{d}\mathbf{x}_j + \text{const.} \tag{34}$$

$$= - \int_j q_j(\mathbf{x}_j) \log\, \frac{q_j(\mathbf{x}_j)}{f_j(\mathbf{x}_j)} \mathrm{d}\mathbf{x}_j + \text{const.} \tag{35}$$

$$= -\mathrm{KL}(q_j(\mathbf{x}_j) || f_j(\mathbf{x}_j)) + \text{const.} \tag{36}$$

The above derivation ends up somewhere rather intuitive, to maximise the lower bound on the evidence with respect to $q_j(\mathbf{x}_j)$ we want to minimise the KL-divergence between the factor $q_j(\mathbf{x}_j$ and the distribution when all *other* factors have been averaged out. Have a look at what the term $f_j(\mathbf{x}_j)$ to see if this makes sense.

As we know that the KL-divergence is always positive and as we are free to choose $q_j(\mathbf{x}_j)$ as we wish we can simply set,

$$q_j(\mathbf{x}_j) = f_j(\mathbf{x}_j) \tag{37}$$

$$\log\, f_j(\mathbf{x}_j) = \int_{\neg j} \underbrace{\prod_{i \neq j} q_i(\mathbf{x}_i)}_{q_{\neg j}(\mathbf{x}_{\neg j})} \log\, p(\mathbf{Y}, \mathbf{X}) \mathrm{d}\mathbf{x}_{\neg j} \tag{38}$$

$$= \mathbb{E}_{q_{\neg j}(\mathbf{x}_{\neg j})} \left[\log\, p(\mathbf{Y}, \mathbf{X})\right] \tag{39}$$

So in order to use the mean-field variational bayes we need to pick the approximate distribution $q(\mathbf{X})$ in such a way that we can compute the expectation above. Now we have derived both variational bayes and the mean field approximation we are ready to move back to our model and work specifically with the Ising model we have defined.

1.6.3 Mean Field Variational Bayes in Ising Model

Now let us formulate the mean field approximation for the Ising model, lets first remind ourselves of the model. We specified a prior of the form,

$$p(\mathbf{x}) = \frac{1}{Z_0} e^{E_0(\mathbf{X})} \tag{40}$$

$$E_0(\mathbf{X}) = \sum_i^N \sum_{j \in (i)} w_{ij} x_i x_j \tag{41}$$

If we look at the term $w_{ij}x_ix_j$ we can see that it will be postive if the latent values are the same and negative otherwise. The larger the value the higher the probability which fits well with our Ising model. The other term we need is the likelihood,

$$p(\mathbf{Y}|\mathbf{X}) = \prod_i p(y_i|x_i) = \frac{1}{Z_1} \prod_i e^{L_i(x_i)}, \tag{42}$$

where the function L_i should give a large value if it is likely that x_i have generated y_i. Now we are ready to formulate our approximate distribution. We will use a full mean-field approximation so we will assume that the approximate distribution over each latent variable is independent,

$$q(\mathbf{x}) = \prod_i q(x_i, \mu_i),$$

where we have introduced μ_i as a variational parameter that parametrises this distribution. In specific μ_i will be $\mathbb{E}_{q_i}[x_i]$. The first thing we need to get is the joint distribution of the model. We will through out the task work in log-space which gives us,

$$\log\, p(\mathbf{x}, \mathbf{y}) = \log\, p(\mathbf{y}|\mathbf{x}) p(\mathbf{x}) \tag{43}$$

$$= \log \left(\prod_i e^{L_i(x_i)} \frac{1}{Z_0} e^{\sum_{j \in \mathcal{N}(i)} w_{ij} x_i x_j} \right) \tag{44}$$

$$= \sum_i \left(L_i(x_i) + \sum_{j \in \mathcal{N}(i)} w_{ij} x_i x_j \right) + \text{const.} \tag{45}$$

As we have choosen a fully factorised approximative distribution $q(\mathbf{x})$ we will compute each expectation in the bound in turn so we want to write up the joint distribution where we only consider one variable. This means,

$$\log\, p(\mathbf{x}, \mathbf{y}) = L_i(x_i) + x_i \sum_{j \in \mathcal{N}(i)} w_{ij} x_j + \text{const.},$$

where we have included all the term over the remaining latent variables in the constant term. We are now ready to compute the expectation to get the approximative posterior,

$$\log\, q_i(x_i) = \log\, f_i(x_i) = \int \prod_{j\neq i} q_j(x_j) \log\, p(\mathbf{x},\mathbf{y}) \mathrm{d}x_{\neg i} \tag{46}$$

$$= \int \prod_{j\neq i} q_j(x_j)(L_i(x_i) + \sum_{k\in\mathcal{N}(i)} w_{ik}x_i x_k + \text{const.})\mathrm{d}x_{\neg i} \tag{47}$$

$$= \underbrace{\int \prod_{j\neq i} q_j(x_j)\mathrm{d}x_{\neg i}}_{=1} L_i(x_i) + \int \prod_{j\neq i} q_j(x_j) \sum_{k\in\mathcal{N}(i)} w_{ik}x_i x_k \mathrm{d}x_{\neg i} + \text{const.} \tag{48}$$

The first integral will compute to one as $q_j(x_j)$ is a distribution. The second term is a bit trickier to deal with so we are going to deal with it on its own.

$$\int \prod_{j\neq i} q_j(x_j) \sum_{k\in\mathcal{N}(i)} w_{ik}x_i x_k \mathrm{d}x_{\neg i} = \int \prod_{j\neq i} q_j(x_j) x_i \sum_{k\in\mathcal{N}(i)} w_{ik}x_k \mathrm{d}x_{\neg i} \tag{49}$$

$$= \int x_i \sum_{k\in\mathcal{N}(i)} w_{ik} \left(\prod_{j\neq i} q_j(x_j)\right) x_k \mathrm{d}x_{\neg i} = x_i \sum_{k\in\mathcal{N}(i)} w_{ik} \int \left(\prod_{j\neq i} q_j(x_j)\right) x_k \mathrm{d}x_{\neg i}. \tag{50}$$

We will now expand the integration over each term and find something rather beautiful,

$$x_i \sum_{k\in\mathcal{N}(i)} w_{ik} \int \left(\prod_{j\neq i} q_j(x_j)\right) x_k \mathrm{d}x_{\neg i} = \tag{51}$$

$$x_i \sum_{k\in\mathcal{N}(i)} w_{ik} \int \left(q_1(x_1)q_2(x_2)\cdot \ldots \cdot q_N(x_N)\right) x_k \mathrm{d}x_1 \mathrm{d}x_2 \cdot \ldots \cdot \mathrm{d}x_N \tag{52}$$

$$= x_i \sum_{k\in\mathcal{N}(i)} w_{ik} \underbrace{\int q_1(x_1)\mathrm{d}x_1}_{=1} \underbrace{\int q_2(x_2)\mathrm{d}x_2}_{=1} \cdot \ldots \cdot \int q_k(x_k)x_k\mathrm{d}x_k \cdot \ldots \cdot \underbrace{\int q_N(x_N)\mathrm{d}x_N}_{=1} \tag{53}$$

$$= x_i \sum_{k\in\mathcal{N}(i)} w_{ik} \int q_k(x_k)x_k\mathrm{d}x_k = x_i \sum_{k\in\mathcal{N}(i)} w_{ik}\mathbb{E}_{q_k(x_k)}[x_k] = \tag{54}$$

$$= x_i \sum_{k\in\mathcal{N}(i)} w_{ik}\mu_k \tag{55}$$

Now we are ready to tidy things up and write out the approximative posterior for x_i by combining our terms,

$$\log\, q_i(x_i) = \log\, f_i(x_i) = L_i(x_i) + x_i \underbrace{\sum_{k\in\mathcal{N}(i)} w_{ik}\mu_k}_{m_i} + \text{const.} \tag{56}$$

$$= L_i(x_i) + x_i \cdot m_i + \text{const.} \tag{57}$$

The expression above does make sense, we have one term which relates to the observations at x_i and one term which relates to the prior term relating the expectations of the nearby latent locations. This means that we can write our approximative distribution as,

$$q(\mathbf{x}) \propto \prod_i q_i(x_i) \propto e^{x_i m_i + L_i(x_i).}$$

We need to make sure that the approximation that we have is an actual distribution therefore making sure that it integrates to 1. In this case this is really simply as $x_i \in [1, -]$ and therefore we can write it up as,

$$\hat{q}_i(x_i = 1) = \frac{1}{q(x_i = 1) + q(x_i = -1)} q(x_i = 1) = \frac{e^{m_i + L_i(1)}}{e^{m_i + L_i(1)} + e^{-m_i + L_i(-1)}} \tag{58}$$

$$= \left\{ \begin{array}{c} \text{Simplification:} \\ \frac{e^a}{e^a + e^b} = \frac{1}{e^{-a}(e^a + e^b)} = \frac{1}{1 + e^{b-a}} \end{array} \right\} \tag{59}$$

$$= \frac{1}{1 + e^{-2m_i - L_i(1) + L_i(-1)}} = \frac{1}{1 + e^{-2\underbrace{(m_i + \frac{1}{2}L_i(1) - \frac{1}{2}L_i(-1))}_{a_i}}} \tag{60}$$

$$= \frac{1}{1 + e^{-2a_i}} = \text{sigm}(2a_i). \tag{61}$$

As the probability for x_i taking value 1 is equal to a sigmoid the probability for the other case is trivial,

$$q_i(x_i = -1) = \text{sigm}(-2a_i)$$

Importantly the proposal distribution is completely defined by its expected value μ_i as a last step we now want to find a way to update this parameter. This is easy to do by going through its definition,

$$\mu_i = \mathbb{E}_{q_i(x_i)}[x_i] = \sum_{x_i \in [1,-1]} x_i q_i(x_i) = (+1) q_i(x_i = 1) + (-1) q_i(x_i = 1) \tag{62}$$

$$= \frac{1}{1 + e^{-2a_i}} - \frac{1}{1 + e^{2a_i}} = \frac{e^{a_i}}{e^{a_i} + e^{-a_i}} - \frac{e^{-a_i}}{e^{-a_i} + e^{a_i}} \tag{63}$$

$$= \frac{e^{a_i} - e^{-a_i}}{e^{a_i} + e^{-a_i}} = \tanh(a_i) = \tanh\left(m_i + \frac{1}{2}(L_i(1) - L_i(-1))\right). \tag{64}$$

So now we are there, we have the approximative distribution for the mean field approximation of an Ising model and we have equation that tells us how to update the parameters of this distribution. Now before we implement this, let us see if it makes sense.

$$q_i(x_i = 1 | \mu_i) = \text{sigm}(2a_i) = \text{sigm}\left(2(m_i + \frac{1}{2}(L_i(1) - L_i(-1)))\right) \tag{65}$$

$$\mu_i = \tanh(a_i) = \tanh\left(m_i + \frac{1}{2}(L_i(1) - L_i(-1))\right) \tag{66}$$

$$m_i = \sum_{j \in \mathcal{N}(i)} w_{ij} \mu_j \tag{67}$$

We are in effect trying to find the probability of a latent variable that can take two different values, this means that a sigmoid function makes perfect sense as an approximative distribution. The update of the variational parameter μ_i is updated as a $\tanh(2ai)$ function this also makes sense as we have $x_i \in [-1, 1]$ we now have an updated which is bounded between those two values. Below we have plotted the two functions,

Except for the asymptotic vaules, does the posterior and the variational update make sense, both are functions of a_i as,

$$a_i = m_i + \frac{1}{2}(L_i(1) - L_i(-1)) = \sum_{j \in \mathcal{N}(i)} w_{ij} \mu_j + \frac{1}{2}(L_i(1) - L_i(-1)).$$

The first term in the above expression relates to the nodes that are neighbours of i. In effect it is a weighting of the expected values of the posteriors of these nodes where the weights w_{ij} are what encodes our prior

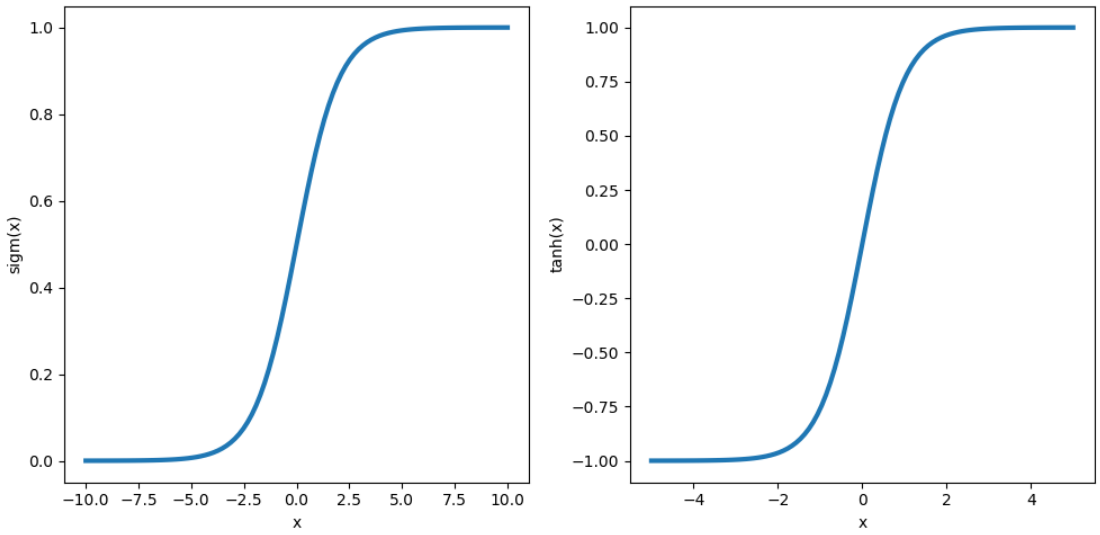

Figure 4: *The above plot shows on the left a sigmoid function and on the right a hyperbolicus tangent function. They look very similar but observe that the sigmoid has its asymptots at 0 and 1 while the tanh is at -1 and 1.*

Algorithm 4 Variational Bayes for Ising Model

procedure MEAN FIELD VARIATIONAL BAYES
  $\boldsymbol{\mu} \leftarrow$ initialise variational distributions
  $\boldsymbol{x} \leftarrow$ initialise latent variables
  for $\tau = 1 \dots T$ **do**
    for $i = 1 \dots N$ **do**
      $m_i^{\tau+1} = \sum_{j \in \mathcal{N}(i)} w_{ij} \mu_j^{\tau} \leftarrow$ compute parameter
      $\mu_i^{\tau+1} = \tanh\left(m_i + \frac{1}{2}(L_i(1) - L_i(-1))\right) \leftarrow$ update variational parameter
  return $q(\mathbf{x})$

assumption in how neighbours interact. As the weights w_{ij} are all positive if all neighbours are in agreement, i.e. have the same sign, then the first term will "push" towards either -1 or $+1$. For simplicity lets assume that all neighours have $\mu_j = 1$ then m_i will be a positive value and vice versa. If it is close to zero that means that the neighbours are all in disagreement **or** that we are very uncertain of their values, i.e. μ_j is close to zero. The second term is a difference between that comes from the likelihood terms, if it is positive this means that it is much more likely that $x_i = 1$ describes the data y_i compared to $x_i = -1$. So for the extremes, if all neighbours are $\mu_j = 1$ and $L_i(1) - L_i(-1) > 0$ then a_i will be a large positive value, i.e. the posterior $q_i(x_i)$ will be large and μ_i will get a value close to one. So these equations do make sense.

1.6.4 Implementation

Now we are ready to finally implement our variational Bayes inference scheme. What we will do is to start off with some initial value for our variational parameter and then we will update each of the distributions in turn. You can try and start with different values and see how it changes the way we reach a solution. The algorithm we should implement is outlined in Algorithm 4.

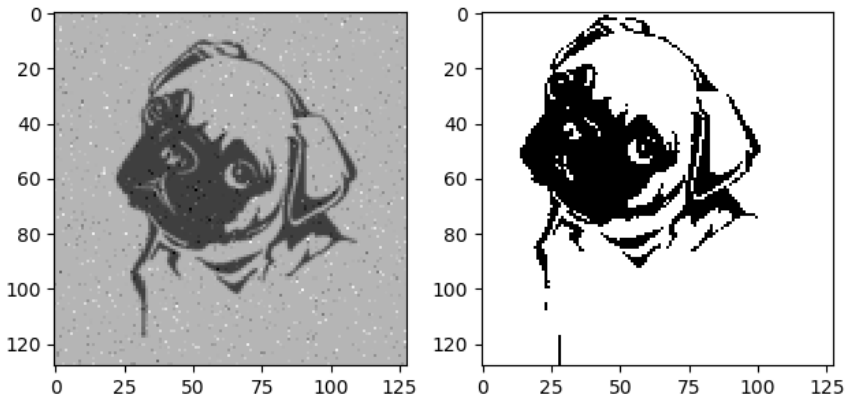

Figure 5: *This figure shows the result of my implementation of variational inference for the image denoising example. As you can see the ising prior cleans up the image rather nicely and we are left with a lovely black and white pug*

Question 6

Implement the algorithm in Algorithm 4 and use it to denoise images as before. My version creates something like can be seen in Figure 5.

Question 7

How does the result of the Variational Bayes method compare to the two previous approaches?

1.7 Image Segmentations

So far we have worked with binary images, this shows the principle as it makes it easy to compute the unary and pairwise terms. However there is nothing stopping us using the same approach for a full colour image. The technique we will look at below is the method that is used in many Micro$oft products for automatic segmentation. The idea is as follows, we will keep the latent variables the same but their semantic meaning will be different, where $+1$ was white and -1 was black they will now indicate if the pixel belongs to the foreground or the background. What we will alter is the likelihoods, what we will do now is keep track of two histograms, one for the foreground colours and one for the background and then we will evaluate our likelihood function as a mean of seeing how likely it is to come from either of the histograms. This is a rather dirty approach but as we are doing computer vision we have lost most of our principles anyway ;-). So what we will do is to create two masks, one that covers the pixels in the image we believe is the foreground and one portion which is the background. Now we will take all the colours in the forground and create a histogram and normalise this. Now we will say that this is the likelihood for the foreground. We will then do the same proceedure for the background. Now you can replace your likelihood evaluation with evaluating this lookup table.

This is a very hacky approach and you can hack even more to produce better results, try to smooth the histogram, try to reduce the number of bins in the histogram. There are lots of things that you can do, play around a bit and see what results that you get.

Question 8

Implement the image segmentation with one of the inference approaches we have gone through, either the sampling based method or the deterministic approach.

1.7.1 Extra material for those interested (will not be marked)

There exists a very very quick method for finding the MAP solution of an MRF called a Graphcut, it is based on a very interesting theorem called the the Ford-Fulkersson theorem [2] or the *max-flow-min-cut* and involves seeing a graph as a flow network. You add two special nodes, a source (s) and a sink (t) water now flows from the source to the sink it turns out that if you keep cutting edges that get saturated with water till you have completely separated the source from the sink you have found something rather interesting. If you now see the source as one label and the sink as the other, each latent variable takes the associated label this turns out to be the MAP solution to the MRF. You can then use this to segment images into foreground and background incredibly fast. The paper that first used this for images is [3] and [4] but it was really popularised by [5]. There are lots of Python implementations for it to try out this webpage keeps links to several of them `http://cmp.felk.cvut.cz/~smidm/python-packages-for-graph-cuts-on-images.html`.

2 Variational Auto-encoder

In this task we will look at one of the recent success stories in machine learning, namely the Variational Auto-encoder [6] or VAE for short. This approach to inference combines several different old ideas together into something that really works and produces very nice results. This part of the coursework will also show you how to work and use an external machine learning packages, in this case Tensorflow from Google [7]. Tensorflow is part of a new trend that started a couple of years back with several different machine learning toolboxes release. They have mostly been focused on neural networks but they are actually useful for general machine learning as well. What they provide is automatic calculations of gradients, when you write your code you it actually never runs. Your code is only the definition of a computational tree and then this is run instead by very efficient implementations. This is very beneficial as one can then perform gradient computations using the chain rule through this tree. One should take care though, as always, abstraction of code always comes with a penalty in performance. Sometimes this can generate some serious nummerical issues so one has to take care. The other thing that is a bit complicated is that as your code is actually never executed you can't really debug it, this is annoying and requires you to think slightly differently. In this task you should run some code written in Tensorflow and try to describe and evaluate the method. First download the implementation of the variational auto-encoder from the repo.

Question 9

Explain briefly how the implementation works, not in terms of code but in terms of the model that it actually implements. What is the main difference between the inference scheme used in the VAE compared to standard variational Bayes? You probably want to have a look at the paper to figure this out.

Question 10

Run the MNIST [8] experiment that is included. Try to visualise what the model has actually learnt. This is a totally open ended question, at this level you should be able to interogate a model to generate results that provides an intuition.

3 Final Thoughts

If you have made it this far you are now done with the assignment part of this unit, well done. You have now done a substantial part of machine learning, and not just applying models, you have *developed* models. You might think that the models are very simple and that the problems that we have solved are trivial but the thing is that to take this type of knowledge up to the level of real data is a much much smaller step compared to the one that you have made during this unit and these two assigments. So now when you are approached by a machine learning problem you are not only able to pick from a set of methods that you know, but you can think, look at the data, and *design* the model that will solve the specif problem that you have at hand. And that is really cool!

Good Luck

References

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

[2] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Journal canadien de mathématiques*, 8(0):399–404, 1956.

[3] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast Approximate Energy Minimization via Graph Cuts. *IEEE Transactions Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001.

[4] Yuri Boykov and M.-P Jolly. Interactive Graph Cuts for Optimal Boundary & Region Segmentation of Objects in N-D Images. In *International Conference on Computer Vision*, pages 105–112. Siemens Corporate Research, IEEE Comput. Soc, 2005.

[5] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. GrabCut: interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics (TOG)*, 23(3):309–314, August 2004.

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

[7] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[8] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.