COMS30007 - Machine Learning

Sampling

Carl Henrik Ek

Week 9

Abstract

In the first set of labs 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 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[1]. 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 the final two labs of this unit we will look at both of these approaches applied to the same problem. This week we will look at the stochastic approximations and next week we will look at the deterministic methods.

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 labs 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 lab 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.

[1] think about how many elements you had in the summation for such a simple problem as the one we looked at in the Evidence lab.

1 The Model

Images are one of the most interesting and easily available sources of data, images contain a lot of information and they can be acquired 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 grey-scale 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[2] 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.

[2]An interesting note of machine learning researchers is that very few comes from a computer science background, much more common are physicists, engineers and of course statisticians. Maybe it is therefore not surprising to see a lot of physics motivated models in use for rather different tasks commpared to what they where initially designed.

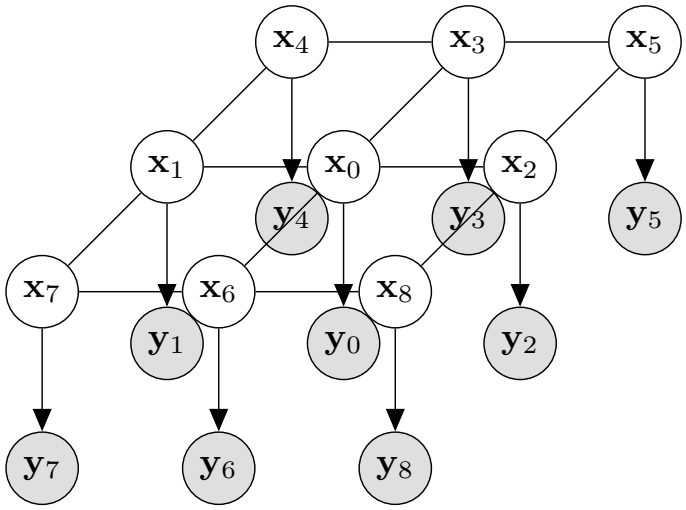

Figure 1: *Above is the graphical model of the MRF we will use for the images. Note that we have connected the latent variables with lines and not arrows, that is because we do not specify these as conditional probabilities but rather joint probabilities.*

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 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 proceed. As a side-note think back on the previous labs where we often 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 actually 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 coordinate-wise gradient approach while the two others are more principled approximations. The lab 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, initialisation etc. so that you get an intuitive understanding for what is going on. Right lets get started!

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 definitely recommend images of pugs, they work great. To make our life a bit easier we use grey-scale images rather than colour. You can use the Imagemagic to convert between colour and grey-scale 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-pepper' noise which flips the pixel values[3].

[3]These noise distributions are very simple, you can also try to code up something more interesting. How about drawing random lines across the image in black or white? When you got your code up and running try to extend the noise models as this will give you a better idea of how the inference actually works.

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).astype('float')
    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.

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?

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 often one of the first approaches you try to get something out of a model and see if it is worth developing a specific taylored inference scheme.

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,

$$\hat{f} = \frac{1}{L}\sum_{l=1}^{L} f(\mathbf{z}^{(l)}) \tag{5}$$

$$\mathbf{z}^{(l)} \sim p(\mathbf{z}). \tag{6}$$

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.

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[4] 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.

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.

[4] `https://en.wikipedia.org/wiki/Manhattan_Project`

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

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,

$$p(x_i | \mathbf{x}_{\neg i}, \mathbf{y}) = \frac{p(\mathbf{x}, \mathbf{y})}{p(\mathbf{x}_{\neg i}, \mathbf{y})}. \tag{7}$$

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

$$p(\mathbf{x}_{\neg i}, \mathbf{y}) = \int p(\mathbf{x}, \mathbf{y}) \mathrm{d}x_i = \sum_{x_i \in [1,-1]} p(x_i, \mathbf{x}_{\neg i}, \mathbf{y}) \tag{8}$$

$$= p(x_i = 1, \mathbf{x}_{\neg i}, \mathbf{y}) + p(x_i = -1, \mathbf{x}_{\neg i}, \mathbf{y}). \tag{9}$$

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

$$p(x_i = 1 | \mathbf{x}_{\neg i}, \mathbf{y}) = \frac{p(x_i = 1, \mathbf{x}_{\neg i}, \mathbf{y})}{p(x_i = 1, \mathbf{x}_{\neg i}, \mathbf{y}) + p(x_i = -1, \mathbf{x}_{\neg i}, \mathbf{y})}, \tag{10}$$

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 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,

$p(x_i = 1 \mid \mathbf{x}_{\neg i}, \mathbf{y})$	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 $\tau = 0 \dots T$ **do**
    for $i = 1 \dots N$ **do**
      $p_i = p(x_i = 1 | \{x_j^{(\tau+1)}\}_{j=1}^{i-1}, \{x_j^{(\tau)}\}_{j=i+1}^{N}, \mathbf{y})$
      $t \sim \text{Uniform}(0, 1)$
      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,

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

$$= \frac{p(y_i|x_i = 1) \prod_{j \neq i} p(y_j|x_j) p(x_i = 1, \mathbf{x}_{\neg i})}{p(y_i|x_i = 1) \prod_{j \neq i} p(y_j|x_j) p(x_i = 1, \mathbf{x}_{\neg i}) + p(y_i|x_i = -1) \prod_{j \neq i} p(y_j|x_j) p(x_i = -1, \mathbf{x}_{\neg i})} \tag{12}$$

$$= \frac{p(y_i|x_i = 1) p(x_i = 1, \mathbf{x}_{\neg i})}{p(y_i|x_i = 1) p(x_i = 1, \mathbf{x}_{\neg i}) + p(y_i|x_i = -1) p(x_i = -1, \mathbf{x}_{\neg i})} \tag{13}$$

$$= \frac{p(y_i|x_i = 1) p(x_i = 1, \mathbf{x}_{\mathcal{N}(i)})}{p(y_i|x_i = 1) p(x_i = 1, \mathbf{x}_{\mathcal{N}(i)}) + p(y_i|x_i = -1) p(x_i = -1, \mathbf{x}_{\mathcal{N}(i)})} \tag{14}$$

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?

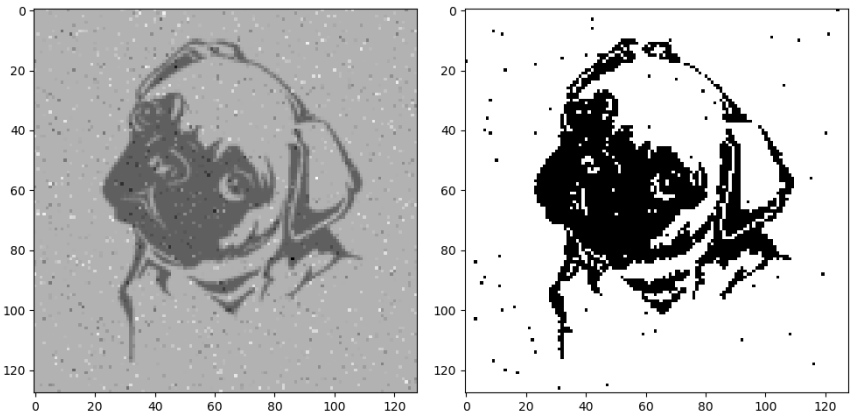

Figure 2: *The result of running the Gibbs sampling approach in the Ising model. The left image is the noisy version while the rightmost is the cleaned up version*

6 Summary

In this lab you have seen how we can approach an intractable computation using an approximative method. The method that you have done can easily be extended to work with colour images but then you have to alter the likelihood function to something a bit more interesting. Next week we will use the same model but perform a deterministic approximation instead.

References

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