Machine Learning in the Physical World

Evidence

Carl Henrik Ek

October 14, 2020

Abstract

Last week we looked at models, we have gone from simple models to figure out if a coin is biased or not to much more complicated infinite dimensional objects to place distributions over the space of functions. We followed the same procedure in each of these tasks, formulate a likelihood and a prior and try to get to the posterior. In this worksheet we are not going to introduce a new model but instead look at the one part of the probabilistic framework that we have so far ignored, the *evidence* or *marginal likelihood*. Hopefully after doing this lab you will see that this object is not just an annoying object that we try to do our best to avoid working with, no its actually the most important of all the probabilistic objects that we have in our arsenal. So lets get aquainted with the evidence.

In this lab we will look at the role the evidence or the marginal likelihood plays in machine learning. We will follow the excellent paper Murray et al., 2005. The evidence is the probability distribution that is left when we have integrated out everything. Say that we have observed a set of data $\mathcal{D}$ and we have built up a model of this parameterised by a set of parameter θ the evidence is,

$$p(\mathcal{D}) = \int p(\mathcal{D}|\theta)p(\theta)\mathrm{d}\theta. \tag{1}$$

This means that the evidence is the distribution over the data space that is created if we average *all" of the possible hypothesis that we have relative to how probable we think that they are. So lets try and get an intuition for this. Lets say that we have a modelling scenario where we have a Gaussian model, and we do not know the mean nor the variance, now to make it simple we have an hypothesis space that only includes three different possible settings of the parameters. This would mean the evidence is an average over these three Gaussians as,

$$p(\mathcal{D}) = \sum_{\boldsymbol{\theta}} p(\mathcal{D}|\theta)p(\theta). \tag{2}$$

In the code below I have written up an example of this and the result can be seen in Figure 1.

Code

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

x = np.linspace(-6,6,200)
pdf1 = norm.pdf(x,0,1)
pdf2 = norm.pdf(x,1,3)
pdf3 = norm.pdf(x,-2.5,0.5)

fig = plt.figure(figsize=(10,5))
ax = fig.add_subplot(111)

ax.plot(x,pdf1,color='r',alpha=0.5)
ax.fill_between(x,pdf1,color='r',alpha=0.3)
ax.plot(x,pdf2,color='g',alpha=0.5)
ax.fill_between(x,pdf2,color='g',alpha=0.3)
ax.plot(x,pdf3,color='b',alpha=0.5)
ax.fill_between(x,pdf3,color='b',alpha=0.3)

pdf4 = 0.3*pdf1 + 0.2*pdf2 + 0.5*pdf3
ax.plot(x, pdf4, color='k', alpha=0.8, linewidth=3.0, linestyle='--')
ax.fill_between(x, pdf4, color='k', alpha=0.5)

# REMOVE THIS
plt.tight_layout()
plt.savefig(path, transparent=True)
return path
```

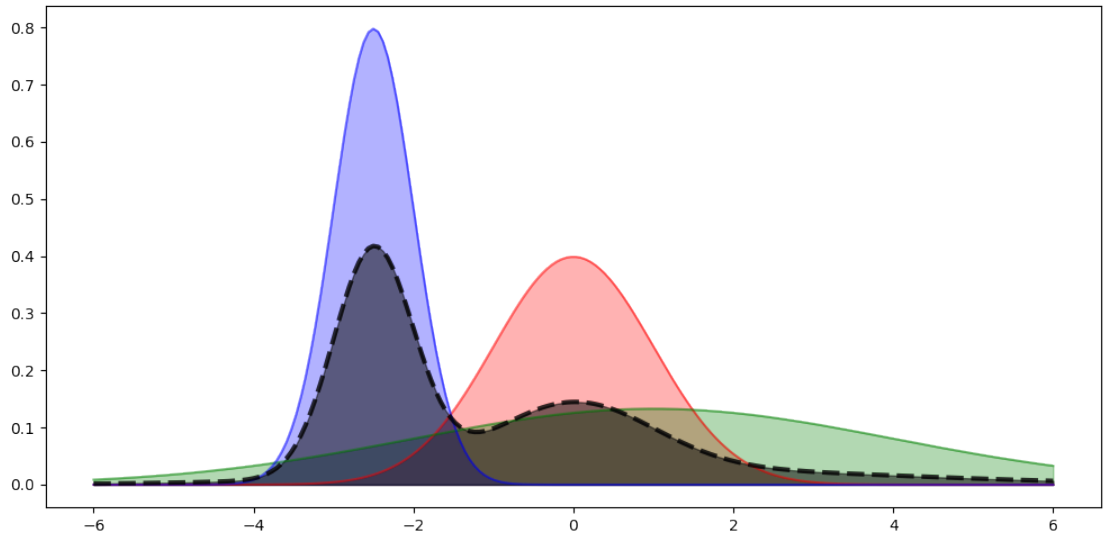

Figure 1: The above figure shows three different parameter settings of a model, *red*, *green* and *blue*. We now marginalise out the parameter according to our belief $p(\theta = \text{red}) = 0.3$, $p(\theta = \text{green}) = 0.2$ and $p(\theta = \text{blue}) = 0.5$ which leads to the evidence in black.

So we should think of the evidence how a model and our beliefs places probability mass over the space where we can later observe data. So now if we would observe some data $\mathbf{Y}$ we can evaluate this under the evidence and say, *what is the evidence that this model is the "right" one?*. Now this becomes very interesting when we have several models. So lets pick another model where instead of Gaussian distribution we have a model using a Laplace distribution giving rise to the evidence plot in Figure 2. So clearly this model and our beliefs in this model places distribution slightly differently across the space.

Code

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import laplace

x = np.linspace(-6,6,200)
pdf1 = laplace.pdf(x,0,1)
pdf2 = laplace.pdf(x,-1,1)
pdf3 = laplace.pdf(x,-2.5,0.5)

fig = plt.figure(figsize=(10,5))
ax = fig.add_subplot(111)

ax.plot(x,pdf1,color='r',alpha=0.5)
ax.fill_between(x,pdf1,color='r',alpha=0.3)
ax.plot(x,pdf2,color='g',alpha=0.5)
ax.fill_between(x,pdf2,color='g',alpha=0.3)
ax.plot(x,pdf3,color='b',alpha=0.5)
ax.fill_between(x,pdf3,color='b',alpha=0.3)

pdf4 = 0.3*pdf1 + 0.2*pdf2 + 0.5*pdf3
ax.plot(x, pdf4, color='k', alpha=0.8, linewidth=3.0, linestyle='--')
ax.fill_between(x, pdf4, color='k', alpha=0.5)

# REMOVE THIS
plt.tight_layout()
plt.savefig(path, transparent=True)
return path
```

So how is this useful, well, so far we have not seen any data, lets say that we now are observing some data $\mathbf{Y}$ which is all centered around -1 as. If we now evaluate the evidence for this data under the two different models we will see that the data is more probably under the model using the Laplace distribution compared to the Gaussian distribution. Simply because the latter model places more of its probability mass just there. This gives us the following relationship,

$$p_{\text{Gaussian}}(\mathcal{D} = \mathbf{Y}) < p_{\text{Laplace}}(\mathcal{D} = \mathbf{Y}), \tag{3}$$

therefore if we would have to choose a model to use to represent this data we would say *There is more or higher evidence for the Laplace model compared to the Gaussian model, we would therefore choose the Laplace model.* This is a really powerful statement as it allows us to test different hypothesis about *model* not just *parameters* of models using this evidence.

This line of reasoning have lead to one of the more famous plots in machine learning, which I like to call the MacKay plot after David Mackay. David was a very influential person in the machine learning community and I would argue that he has a lot to do with the prominent position the UK has played in the development of this field. Slightly outside this topic he also wrote an excellent book on global warming called Without

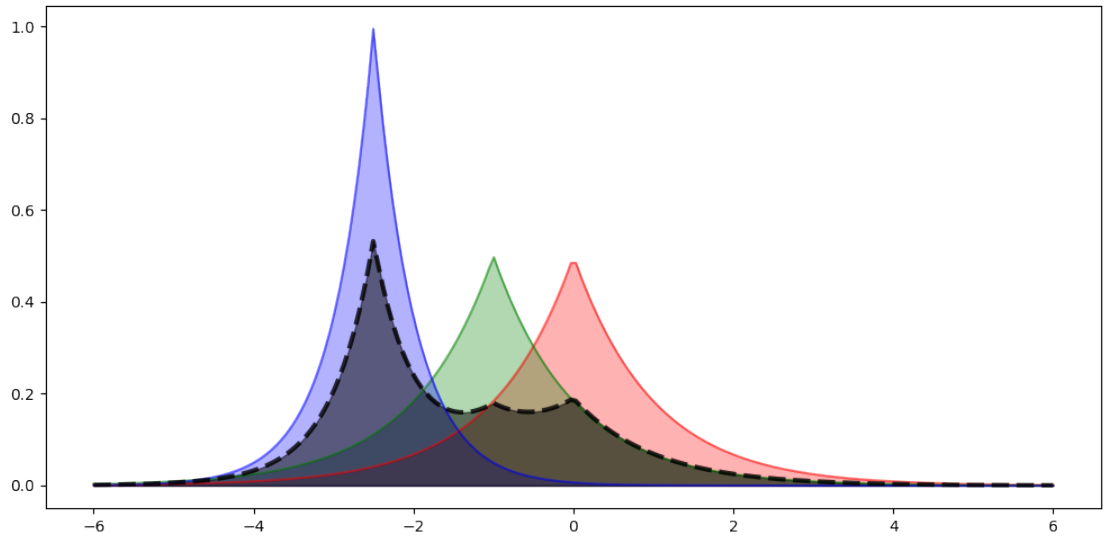

Figure 2: The evidence computed for a slightly different model where we have Laplace distributions that we do not know the parameters of.

Hot Air which is a fantastic read about the challenges of energy. But back on track, David put this Figure 3 of the evidence in his PhD thesis.

The idea of the plot is that if you sort the data-space such that the simpler the data is the further left it is. Now this would mean that the "simpler" the model is then it will place probability mass further left. Now if we are now to choose model between the three above we can use the argument of Occams Razor which simply states that you should *pick the simplest model possible that explains your data*. Now as probability distributions have to sum to 1 the model that has the highest probability when evaluated at the data will thefore have to have placed less probability mass elsewhere and therefore contain less explanations. Therefore picking the model with the maximum evidence would therefore be choosing the model according to Occams Razor. In Figure 3 model $\mathcal{M}_1$ is too simple and $\mathcal{M}_3$ is too complicated and explains too many things while $\mathcal{M}_2$ is just right and this is exactly what the evidence would encode as $p_{\mathcal{M}_2}(\mathcal{D} = \mathcal{D}_0)$ would be higher than both $p_{\mathcal{M}_1}(\mathcal{D} = \mathcal{D}_0)$ and $p_{\mathcal{M}_3}(\mathcal{D} = \mathcal{D}_0)$. Think about this, does it make sense?

Hopefully you think that the argument above makes sense and see that this can be quite a powerful technique. I think its a really strong argumentations and a nice codification of Occam's Razor. Say that you meet someone that believes that the earth is flat, most likely they will present you with evidence that they think supports their theory and most likely[1] the examples that they will give and the explanations that they give will fit. But still, you don't believe them, you think that their arguments are so convoluted, so complicated so really you just go about your way believing the earth is a sphere and then move on. So really, you decided that their model didn't fit because the model that they choose was too *complicated* so you choose another model that was *simpler* but importantly both provided an equivivalent explanation of the data. This means you have acted according to Occams Razor. However, if this is the case, why does anyone believe that the earth is flat[2] simply because the notion of simple is not universal. We all have a different idea of what is simple, and because of this the implementation of Occam's Razor is subjective. You first have to define what simple is in order to use the concept, again, *there is no free-lunch*. What we will now do for the rest of the worksheet is to implement a scenario where we can actually test these concepts and make our understanding of this a bit more clear. We will do so by repeating the experiments of Murray et al., 2005. This paper is a really good read, what I especially like about it is that it doesn't provide an answer, it just raises more

[1] otherwise they will be a bit silly

[2] or that Bristol Rovers is a good football team

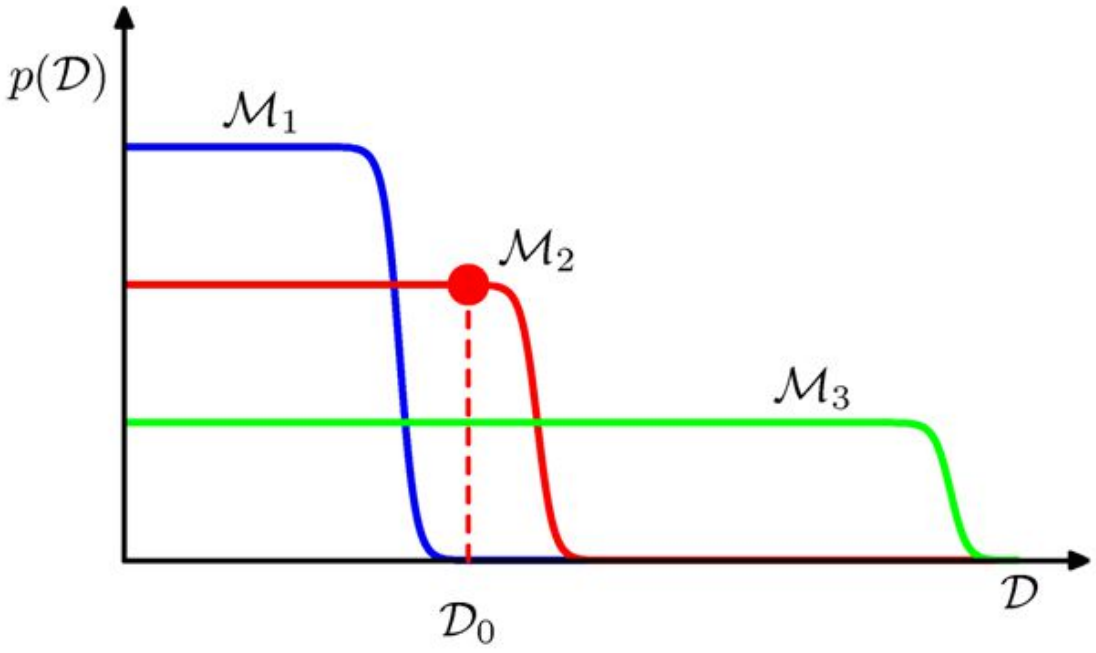

Figure 3: This is the famous Mackay plot, the idea is that if you compute the evidence under three different models, the green, red and blue. The data that you actually observe is $\mathcal{D}_0$ which is evaluated under the three different distributions.

questions.

1 A Note on the Evidence and Bayesian Occam's Razor

The reason that the module initially have focused on conjugate models is because we wanted to avoid computing Baye's Rule to reach the posterior distribution. Conjugacy allowed us to avoid computing the denominator and simply multiply prior and likelihood, then identify the terms to be able to normalise the posterior. For most models this is not possible and we are required to actually perform the full computation and solve an often intractable integral to reach the posterior distribution. Now the object that we want to look at is the evidence so avoiding computation of it is rather pointless. Because of this we are going to choose a discrete data domain whos cardinality is so small so that we can actually evaluate the evidence for all possible data-sets. We will start off by first creating the data-set then we will move on and create a set of models that we can compute the evidence under.

1.1 Data

Consider a very simple data domain $\mathcal{D} = \{y^i\}_{i=1}^9$ where $y^i \in \{-1, 1\}$. This data is structured according to a grid whos locations can be parametrised by $\mathcal{X} = \{\mathbf{x}^i\}_{i=1}^9$ where $\mathbf{x}^i = (\{-1, 0, +1\}, \{-1, 0, +1\})$. This means that our data domain $\mathcal{D}$ contains $2^9 = 512$ different elements which is small enough for us to reason about but still complicated enough that it requires a sensible model.

We will now generate all possible data-sets so that we can evaluate the evidence. The code below will generate a list that you can iterate through with all the possible data points.

Code

```
import itertools as it

def generate_data(N=3):
    D2 = np.array(list(it.product([-1,1],repeat=N*N)))
    D = [];
    for i in range(0,len(D2)):
        d = D2[i]
        D.append(d.reshape([N,N]))
        x = [];
    for i in range(-(N-2),N-1):
        for j in range(-(N-2),N-1):
            x.append(np.array([float(i),float(j)]))
    return (D,x)
```

1.2 Models

Given the data defined above we wish to create a model, i.e. something that will explain the statistical variations that are possible in $\mathcal{D}$. The simplest model that (I) can think of is something that simply takes all its probability mass and places it uniformly over the whole data space,

$$p(\mathcal{D}|M_0, \boldsymbol{\theta}_0) = \frac{1}{512}. \tag{4}$$

The first model Eq. 4 does not take any parameters at all which means it has no flexibility and uses no information about $\mathcal{D}$ except for its cardinality. We can use what we know about the data in order to specify something slightly more representative. If we assume that each y^i are independent we can factorise the model into 9 separate models,

$$p(\mathcal{D}|M_1, \boldsymbol{\theta}_1) = \prod_{n=1}^{9} p(y^i|M_1, \boldsymbol{\theta}_1), \tag{5}$$

where θ_i^j means the j:th element of the parameter vector for the t:th model. Each model can be expressed using an exponential function which relates the vaule y^i to its location $\mathbf{x}^i$,

$$p(\mathcal{D}|M_1, \boldsymbol{\theta}_1) = \prod_{n=1}^{9} \frac{1}{1 + e^{-y^n \theta_1^1 x_1^n}}, \tag{6}$$

We can continue to add more parameters and create further models,

$$p(\mathcal{D}|M_2, \boldsymbol{\theta}_2) = \prod_{n=1}^{9} \frac{1}{1 + e^{-y^n (\theta_2^1 x_1^n + \theta_2^2 x_2^n)}} \tag{7}$$

$$p(\mathcal{D}|M_3, \boldsymbol{\theta}_3) = \prod_{n=1}^{9} \frac{1}{1 + e^{-y^n (\theta_3^1 x_1^n + \theta_3^2 x_2^n + \theta_3^3)}}, \tag{8}$$

Now we can implement the three different models such that we can return the probability for a specific data-point.

Code

```
def model0(theta,x,y):
    return 1.0/(pow(2.,len(x)))

def model1(theta,x,y):
    model = 1.0
    for i in range(len(x)):
        model *= 1.0/(1+exp(-y[i]*theta[0]*x[i][0]))
    return model

def model2(theta,x,y):
    model = 1.0
    for i in range(len(x)):
        model *= 1.0/(1+exp(-y[i]*(theta[0]*x[i][0]+theta[1]*x[i][1])))
    return model

def model3(theta,x,y):
    model = 1.0
    for i in range(len(x)):
        model *= 1.0/(1+exp(-y[i]*(theta[0]*x[i][0]+theta[1]*x[i][1]+theta[2])))
    return model
```

Now we have both the models and the data and its time to move on to the actual computation of the evidence.

1.3 Evidence

The evidence of a model M_i is the distribution $p(\mathcal{D}|M_i)$. This distribution tells us how and where the model spreads its probability mass. Occam's razor can be interpreted in terms of the evidence such as we should choose a model which places most of its mass where we will see data and as little as possible elsewhere. In the previous section we have defined a small simple data domain $\mathcal{D}$ and we will now evaluate where the different models defined above places their probability mass.

In order to "reach" the evidence of a model we need to first remove the dependency of the variable $\boldsymbol{\theta}$. This can be done by marginalising out the parameters from the model,

$$p(\mathcal{D}|M_i) = \int_{\forall \boldsymbol{\theta}} p(\mathcal{D}|M_i, \boldsymbol{\theta}) p(\boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta}. \tag{9}$$

The marginalisation above requires one more object that we haven't seen before $p(\boldsymbol{\theta}|M_i)$. This is the prior over the parameters of the model. Being Bayesian implies that you need to take uncertainty into account in all steps of your calculations this is true for the data but also true for the parameters. As we do not really know much at all about the parameters we would like to be very uncertain and allow for a large range of possible values of $\boldsymbol{\theta}$. One prior would be to choose a simple Gaussian with zero mean and a very large variance,

$$\begin{aligned} p(\boldsymbol{\theta}|M_i) &= \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \\ \boldsymbol{\mu} &= \mathbf{0} \\ \boldsymbol{\Sigma} &= \sigma^2 \boldsymbol{I} \\ \sigma^2 &= 10^3 \end{aligned} \tag{10}$$

Now when we have defined the prior $p(\boldsymbol{\theta})$ we just need to perform the marginalisation in Eq. 9 to be able to evaluate the evidence. However, this integration is rather tricky to do analytically which means that we

will here use an approximate integral using a naive Monte Carlo approach,

$$p(\mathcal{D}|M_i) \approx \frac{1}{S}\sum_{s=1}^{S} p(\mathcal{D}|M_i, \boldsymbol{\theta}^s), \tag{11}$$

$$\boldsymbol{\theta}^s \sim p(\boldsymbol{\theta}|M_i) \tag{12}$$

where s indexes the samples from the prior of the parameters. Do not worry too much about this proceedure right now, hopefully there was enough insight into sampling during the lecture but as I said then, approximative inference is not the topic of this course. Right now, just see this as a black-box proceedure that we are doing.

The code that you need in order to generate the evidence for the different models is

Code

```
# generate the parameter vector for the samples
def generate_parameters(N,d,mu,sigma):
    return sigma*np.random.randn(N,d)+mu

# generate the evidence
def compute_evidence(y,x,theta,model):
    evidence = 0.0
    for i in range(len(theta)):
        evidence += model(theta[i],x,y)
    return evidence/len(theta)
```

Now we have our models, we have our data and we have an approach to reach the evidence for each model. It is now time to run some experiments and see where this leads to. If you have set things up correctly you should be able to run something similar to,

Code

```
N = 3;
nr_samples = pow(10,2)
sigma = pow(10,1.5)
mu = 0
[D, x] = generate_data(N)
[theta, theta_prior] = generate_parameters(nr_samples,3,mu,sigma)

evidence = np.zeros([4,len(D)])

for i in range(len(D)):
    evidence[0,i] = compute_evidence(D[i].ravel(),x,theta,model0)
    evidence[1,i] = compute_evidence(D[i].ravel(),x,theta,model1)
    evidence[2,i] = compute_evidence(D[i].ravel(),x,theta,model2)
    evidence[3,i] = compute_evidence(D[i].ravel(),x,theta,model3)
```

What we now have is the evidence compute under each of the different models. We can now look at how the distribute probability mass over $\mathcal{D}$. The big question is how to sort the data-set, what order should the data have? One thing that you can try is to sort the data according to one model and plot all model using that one. You can get the indices that sorts an array by using `np.argsort()`. So try something like this,

Code

```
index = np.argsort(evidence[1,:])
ax.plot(evidence[0,index], 'r')
ax.plot(evidence[1,index], 'g')
ax.plot(evidence[2,index], 'b')
ax.plot(evidence[3,index], 'k')
```

When you have got everything running, go back to the arguments we did to motivate the evidence. Does it make sense, do you feel that this is supported in the experiments?

2 Summary

Hopefully you have reached the end of this lab and quite possibly you are a bit confused at this point. What am I actually supposed to have learnt from this? Lets go and think about it from the start of the course, we argued that it is impossible to make any type of learning without making assumptions or having beliefs. Now we have just taken this to its extreme and made an argument that this is also true for Occam's Razor, this is truly a subjective argument as it relies on the concept of simple. The second argument is looking at the plots seeing how the different models distribute their probablity mass, some models represent certain types of data well and seeing that when building models it is alway a choice, if you are good at something you always pay the price for being bad at something else, the important thing is therefore to choose the model which is good at the relevant thing, the thing that you are interested in.

References

Murray, Iain and Zoubin Ghahramani (Aug. 2005). *A note on the evidence and Bayesian Occam's razor.* Tech. rep. GCNU-TR 2005-003.

Machine Learning in the Physical World

Approximate Inference

Carl Henrik Ek

October 19, 2020

Abstract

In the first two worksheets we looked at how we can create models that allows us to parametrise and factorise a distribution over the observed data domain. 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 this worksheet we will pick a model and derive a set of simple stochastic inference mechanisms and a deterministic update.

Information

This is an optional worksheet that you can look at if you want to do hands-on the material that we skimmed through in the lecture on approximate inference and latent variables. It is not a core part of the course but it is something that is hard to avoid still. Most of the time we can use reliable black-box methods for inference. They will most likely be rather inefficient as they are general and you can gain a lot by tailor-made inference mechanisms.

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,

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

[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 worksheet.

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.

0.1 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,

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

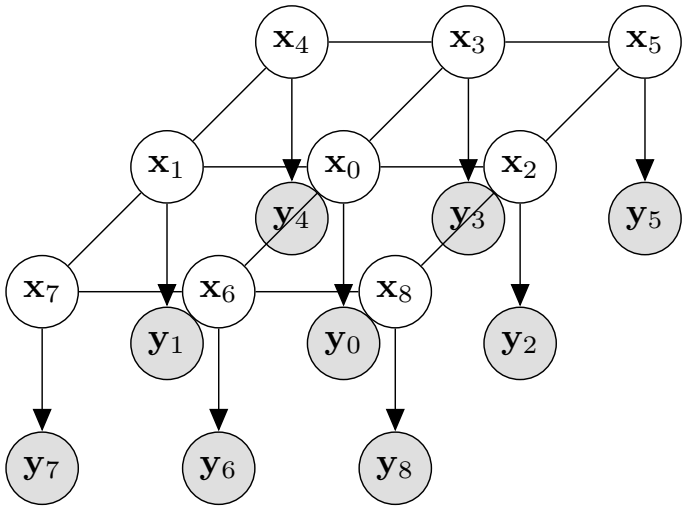

Figure 1: *Above is the graphical model of the MRF we will use for the images. Note that we have connected the latent variables with lines and not arrows, that is because we do not specify these as conditional probabilities but rather joint probabilities.*

We 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.

0.2 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.

[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.

We will now proceed to look at three different approaches to inferring the latent pixel values in the above modell first method is just a simple coordinate-wise gradient approach, the second one is a more principled stochastic approximations while the last is a deterministic approximation. The experiments should be fairly straight forward to code up but what I want you to try and do, as this is 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!

0.3 Data

First we need some data to work with, you can use any image that you want as a starting point. To make our life a bit easier we use grey-scale images rather than colour as this simplifies our likelihood function. You can use the `Imagemagick` 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 = 0.2
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 Sampling

Now we will proceed with a set of stochastic inference mechanisms. The first one *Iteratice Conditional Modes* is really just here for a baseline so that you can see that the MCMC method that we implement after provides quite some benefits.

1.1 Iterative Conditional Modes (ICM)

The first approach we should take is something called Iterative Conditional Modes. 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.

1.2 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. Gibbs sampling is often quite easy to implement so

Algorithm 1 Iterative Conditional Modes for Ising Model

```
procedure IMAGE DENOISING WITH ICM
    x ← initialisation
    for τ = 1 ... T do
        for i = 1 ... N do
            if p(x_i = 1, x_¬i, y) > p(x_i = −1, x_¬i, y) then
                x_i = 1
            else
                x_i = −1
    return x
```

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.

1.2.1 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.

1.2.2 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.

1.2.3 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

1.2.4 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,

$p(x_i = 1\|\mathbf{x}_{\neg i}, \mathbf{y})$	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 we

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 $\mathbf{x}^{(T)}$

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.

Reflections

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)`.

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?

1.3 Summary

In the first part of this worksheet 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.

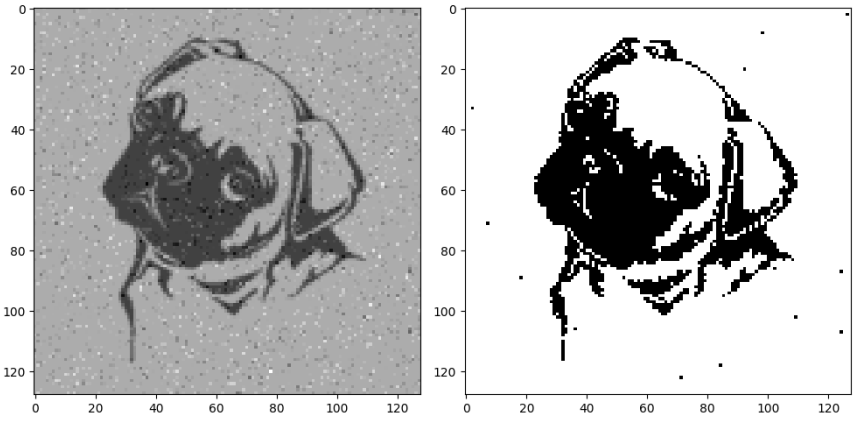

Figure 2: *The result of running the Gibbs sampling approach in the Ising model. The left image is the noisy version while the rightmost is the cleaned up version*

2 Variational 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.

2.1 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,

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

What this means is that we have integrated out everything that we have beliefs over, all the random variables in the model. Now the remaining variables we want to make a point-estimate over. As we know the evidence is often intractable to compute we need to get some other way around this problem. Lets see what we can do,

$$\log p(\mathbf{Y}) = \log \int p(\mathbf{Y}, \mathbf{X}) \mathrm{d}\mathbf{X} = \log \int p(\mathbf{X}|\mathbf{Y}) p(\mathbf{Y}) \mathrm{d}\mathbf{X} \tag{15}$$

$$= \log \int \frac{q(\mathbf{X})}{q(\mathbf{X})} p(\mathbf{X}|\mathbf{Y}) p(\mathbf{Y}) \mathrm{d}\mathbf{X}. \tag{16}$$

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 the curve see Figure 3.

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

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

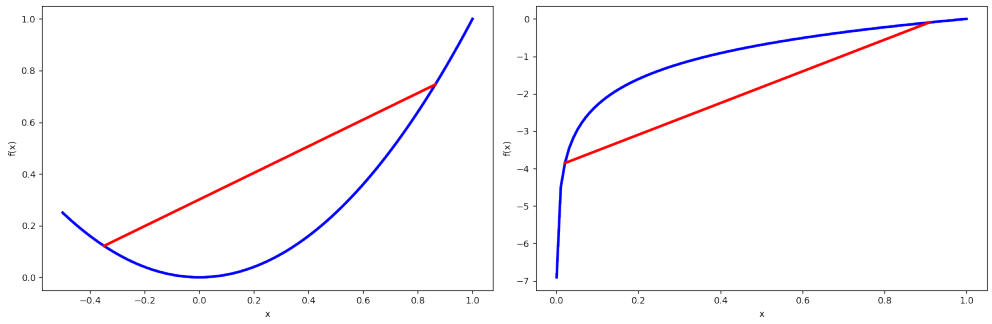

Figure 3: *The plot on the left shows a convex function in blue and line connecting two of the points on the function. The Jensen inequality implies that if you "move" a point along the red line it will always be above the blue. On the right the blue function is a logarithm and we can see that the reverse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

of a convex function, the expectation of the function will always be an upper bound on the function applied to the expectations,

$$\mathbb{E}[f(x)] \geq f(\mathbb{E}[x]) \tag{17}$$

$$\int f(x)p(x)\mathrm{d}x \geq f\left(\int xp(x)\mathrm{d}x\right) \tag{18}$$

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,

$$\int \log(x)p(x)\mathrm{d}x \leq \log\left(\int xp(x)\mathrm{d}x\right). \tag{19}$$

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

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

$$\geq \int q(\mathbf{X})\log\frac{p(\mathbf{X}|\mathbf{Y})p(\mathbf{Y})}{q(\mathbf{X})}\mathrm{d}\mathbf{X} \tag{21}$$

$$= \int q(\mathbf{X})\log\frac{p(\mathbf{X}|\mathbf{Y})}{q(\mathbf{X})}\mathrm{d}\mathbf{X} + \int q(\mathbf{X})\mathrm{d}\mathbf{X}\log\, p(\mathbf{Y}) \tag{22}$$

$$= -\mathrm{KL}\left(q(\mathbf{X})||p(\mathbf{X}|\mathbf{Y})\right) + \log\, p(\mathbf{Y}). \tag{23}$$

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. This is a measure of "simiarity" between probability distributions. It is not a metric, its for example not symmetric, but importantly it is only 0 if the two distributions are the same and positive in all other cases[5]. 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

[5]its quite easy to check this if you are interested

model. Therefore lets try to minimise the KL-divergence between the two distributions.

$$\mathrm{KL}(q(\mathbf{X})||p(\mathbf{X}|\mathbf{Y})) = \int q(\mathbf{X}) log \frac{q(\mathbf{X})}{p(\mathbf{X}|\mathbf{Y})} \mathrm{d}\mathbf{X} \tag{24}$$

$$= \int q(\mathbf{X}) log \frac{q(\mathbf{X})}{p(\mathbf{X}, \mathbf{Y})} \mathrm{d}\mathbf{X} + log\ p(\mathbf{Y}) \tag{25}$$

$$= H(q(\mathbf{X})) - \mathbb{E}_{q(\mathbf{X})} \left[log\ p(\mathbf{X}, \mathbf{Y}) \right] + log\ p(\mathbf{Y}). \tag{26}$$

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,

$$log\ p(\mathbf{Y}) = \mathrm{KL}(q(\mathbf{X})||p(\mathbf{X}|\mathbf{Y})) + \underbrace{\mathbb{E}_{q(\mathbf{X})} \left[log\ p(\mathbf{X}, \mathbf{Y}) \right] - H(q(\mathbf{X}))}_{\text{ELBO}} \tag{27}$$

$$\geq \mathbb{E}_{q(\mathbf{X})} \left[log\ p(\mathbf{X}, \mathbf{Y}) \right] - H(q(\mathbf{X})) = \mathcal{L}(q(\mathbf{X})). \tag{28}$$

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.

In the next part we will derive a specific family of approximations often referred to as mean-field approximations[6]. They are often not particularly exact but they do work in most cases.

2.2 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.

$$\mathcal{L}(q) = \int q(\mathbf{X}) \log \frac{p(\mathbf{Y}, \mathbf{X})}{q(\mathbf{X})} \mathrm{d}\mathbf{X} = \int \prod_i q_i(\mathbf{x}_i) \log \frac{p(\mathbf{Y}, \mathbf{X})}{\prod_k q_k(\mathbf{x}_k)} \mathrm{d}\mathbf{X} \tag{29}$$

$$= \int \prod_i q_i(\mathbf{x}_i) \left(\log p(\mathbf{Y}, \mathbf{X}) - \sum_k \log q_k(\mathbf{x}_k) \right) \tag{30}$$

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,

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

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

[6] Again these are models that initially was suggested in physics to model phase transitions first described by Pierre Curie. If you read the literature on Variational Bayes you will often hear the ELBO referred to as the variational free energy, and the first terms the bound as *energy* as the second term corresponds to the *entropy* of the approximating distribution. So don't forget your theromdynamics.

$$\mathcal{L}(q) = \int \prod_i q_i(\mathbf{x}_i) \left(\log p(\mathbf{Y}, \mathbf{X}) - \sum_k \log q_k(\mathbf{x}_k) \right) \tag{31}$$

$$= \int_j \int_{\neg j} q_j(\mathbf{x}_j) \prod_{i \neq j} q_i(\mathbf{x}_i) \left(\log\, p(\mathbf{X}, \mathbf{Y}) - \sum_k \log\, q_k(\mathbf{x}_k) \right) \mathrm{d}\mathbf{x}_{\neg j} \mathrm{d}\mathbf{x}_j \tag{32}$$

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

$$- \int_j q_j(\mathbf{x}_j) \int_{\neg j} \prod_{i \neq j} q(\mathbf{x}_i) \left(\log\, q_j(\mathbf{x}_j) + \sum_{k \neq j} \log\, q_k(\mathbf{x}_k) \right) \mathrm{d}\mathbf{x}_{\neg j} \mathrm{d}\mathbf{x}_j \tag{33}$$

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

$$- \int_j q_j(\mathbf{x}_j) \left(\log\, q_j(\mathbf{x}_j) \underbrace{\int_{\neg j} \prod_{i \neq j} q_i(\mathbf{x}_i) \mathrm{d}\mathbf{x}_{\neg j}}_{=1} + \underbrace{\int_{\neg j} \prod_{i \neq j} q_i(\mathbf{x}_i) \sum_{k \neq j} \log\, q_k(\mathbf{x}_k) \mathrm{d}\mathbf{x}_{\neg j}}_{\text{constant w.r.t. } q_j} \right) \tag{34}$$

$$= \int_j q_j(\mathbf{x}_j) \log\, f_j(\mathbf{x}_j) \mathrm{d}\mathbf{x}_j - \int_j q_j(\mathbf{x}_j) \log\, q_j(\mathbf{x}_j) \mathrm{d}\mathbf{x}_j + \text{const.} \cdot \underbrace{\int_j q_j(\mathbf{x}_j) \mathrm{d}\mathbf{x}_j}_{=1} \tag{35}$$

$$= \int_j q_j(\mathbf{x}_j) \log\, \frac{f_j(\mathbf{x}_j)}{q_j(\mathbf{x}_j)} \mathrm{d}\mathbf{x}_j + \text{const.} \tag{36}$$

$$= - \int_j q_j(\mathbf{x}_j) \log\, \frac{q_j(\mathbf{x}_j)}{f_j(\mathbf{x}_j)} \mathrm{d}\mathbf{x}_j + \text{const.} \tag{37}$$

$$= -\mathrm{KL}(q_j(\mathbf{x}_j) || f_j(\mathbf{x}_j)) + \text{const.} \tag{38}$$

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,

$$q_j(\mathbf{x}_j) = f_j(\mathbf{x}_j) \tag{39}$$

$$\log\, f_j(\mathbf{x}_j) = \int_{\neg j} \underbrace{\prod_{i \neq j} q_i(\mathbf{x}_i)}_{q_{\neg j}(\mathbf{x}_{\neg j})} \log\, p(\mathbf{Y}, \mathbf{X}) \mathrm{d}\mathbf{x}_{\neg j} \tag{40}$$

$$= \mathbb{E}_{q_{\neg j}(\mathbf{x}_{\neg j})} \left[\log\, p(\mathbf{Y}, \mathbf{X}) \right] \tag{41}$$

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.

2.3 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,

$$p(\mathbf{x}) = \frac{1}{Z_0} e^{E_0(\mathbf{X})} \tag{42}$$

$$E_0(\mathbf{X}) = \sum_i^N \sum_{j \in (i)} w_{ij} x_i x_j \tag{43}$$

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 likelhood,

$$p(\mathbf{Y}|\mathbf{X}) = \prod_i p(y_i|x_i) = \frac{1}{Z_1} \prod_i e^{L_i(x_i)}, \tag{44}$$

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,

$$\log p(\mathbf{x}, \mathbf{y}) = \log p(\mathbf{y}|\mathbf{x}) p(\mathbf{x}) \tag{45}$$

$$= \log \left(\prod_i e^{L_i(x_i)} \frac{1}{Z_0} e^{\sum_{j \in \mathcal{N}(i)} w_{ij} x_i x_j} \right) \tag{46}$$

$$= \sum_i \left(L_i(x_i) + \sum_{j \in \mathcal{N}(i)} w_{ij} x_i x_j \right) + \text{const.} \tag{47}$$

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,

$$\log q_i(x_i) = \log f_i(x_i) = \int \prod_{j \neq i} q_j(x_j) \log p(\mathbf{x}, \mathbf{y}) \mathrm{d}x_{\neg i} \tag{48}$$

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

$$= \underbrace{\int \prod_{j \neq i} q_j(x_j) \mathrm{d}x_{\neg i}}_{=1} L_i(x_i) + \int \prod_{j \neq i} q_j(x_j) \sum_{k \in \mathcal{N}(i)} w_{ik} x_i x_k \mathrm{d}x_{\neg i} + \text{const.} \tag{50}$$

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.

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

$$= \int x_i \sum_{k\in\mathcal{N}(i)} w_{ik} \left(\prod_{j\neq i} q_j(x_j)\right) x_k \mathrm{d}x_{\neg i} = x_i \sum_{k\in\mathcal{N}(i)} w_{ik} \int \left(\prod_{j\neq i} q_j(x_j)\right) x_k \mathrm{d}x_{\neg i}. \tag{52}$$

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

$$x_i \sum_{k\in\mathcal{N}(i)} w_{ik} \int \left(\prod_{j\neq i} q_j(x_j)\right) x_k \mathrm{d}x_{\neg i} = \tag{53}$$

$$x_i \sum_{k\in\mathcal{N}(i)} w_{ik} \int \left(q_1(x_1)q_2(x_2)\cdot\ldots\cdot q_N(x_N)\right) x_k \mathrm{d}x_1 \mathrm{d}x_2 \cdot\ldots\cdot \mathrm{d}x_N \tag{54}$$

$$= x_i \sum_{k\in\mathcal{N}(i)} w_{ik} \underbrace{\int q_1(x_1)\mathrm{d}x_1}_{=1} \underbrace{\int q_2(x_2)\mathrm{d}x_2}_{=1} \cdot\ldots\cdot \int q_k(x_k)x_k\mathrm{d}x_k \cdot\ldots\cdot \underbrace{\int q_N(x_N)\mathrm{d}x_N}_{=1} \tag{55}$$

$$= x_i \sum_{k\in\mathcal{N}(i)} w_{ik} \int q_k(x_k)x_k\mathrm{d}x_k = x_i \sum_{k\in\mathcal{N}(i)} w_{ik} \mathbb{E}_{q_k(x_k)}[x_k] = \tag{56}$$

$$= x_i \sum_{k\in\mathcal{N}(i)} w_{ik}\mu_k \tag{57}$$

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

$$\log\, q_i(x_i) = \log\, f_i(x_i) = L_i(x_i) + x_i \underbrace{\sum_{k\in\mathcal{N}(i)} w_{ik}\mu_k}_{m_i} + \text{const.} \tag{58}$$

$$= L_i(x_i) + x_i \cdot m_i + \text{const.} \tag{59}$$

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,

$$\hat{q}_i(x_i = 1) = \frac{1}{q(x_i = 1) + q(x_i = -1)} q(x_i = 1) = \frac{e^{m_i + L_i(1)}}{e^{m_i + L_i(1)} + e^{-m_i + L_i(-1)}} \tag{60}$$

$$= \left\{ \begin{array}{c} \text{Simplification:} \\ \frac{e^a}{e^a + e^b} = \frac{1}{e^{-a}(e^a + e^b)} = \frac{1}{1 + e^{b-a}} \end{array} \right\} \tag{61}$$

$$= \frac{1}{1 + e^{-2m_i - L_i(1) + L_i(-1)}} = \frac{1}{1 + e^{-2\underbrace{(m_i + \frac{1}{2}L_i(1) - \frac{1}{2}L_i(-1))}_{a_i}}} \tag{62}$$

$$= \frac{1}{1 + e^{-2a_i}} = \text{sigm}(2a_i). \tag{63}$$

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,

$$\mu_i = \mathbb{E}_{q_i(x_i)}[x_i] = \sum_{x_i \in [1,-1]} x_i q_i(x_i) = (+1)q_i(x_i = 1) + (-1)q_i(x_i = 1) \tag{64}$$

$$= \frac{1}{1 + e^{-2a_i}} - \frac{1}{1 + e^{2a_i}} = \frac{e^{a_i}}{e^{a_i} + e^{-a_i}} - \frac{e^{-a_i}}{e^{-a_i} + e^{a_i}} \tag{65}$$

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

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.

$$q_i(x_i = 1|\mu_i) = \text{sigm}(2a_i) = \text{sigm}\left(2(m_i + \frac{1}{2}(L_i(1) - L_i(-1)))\right) \tag{67}$$

$$\mu_i = \tanh(a_i) = \tanh\left(m_i + \frac{1}{2}(L_i(1) - L_i(-1))\right) \tag{68}$$

$$m_i = \sum_{j \in \mathcal{N}(i)} w_{ij}\mu_j \tag{69}$$

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,

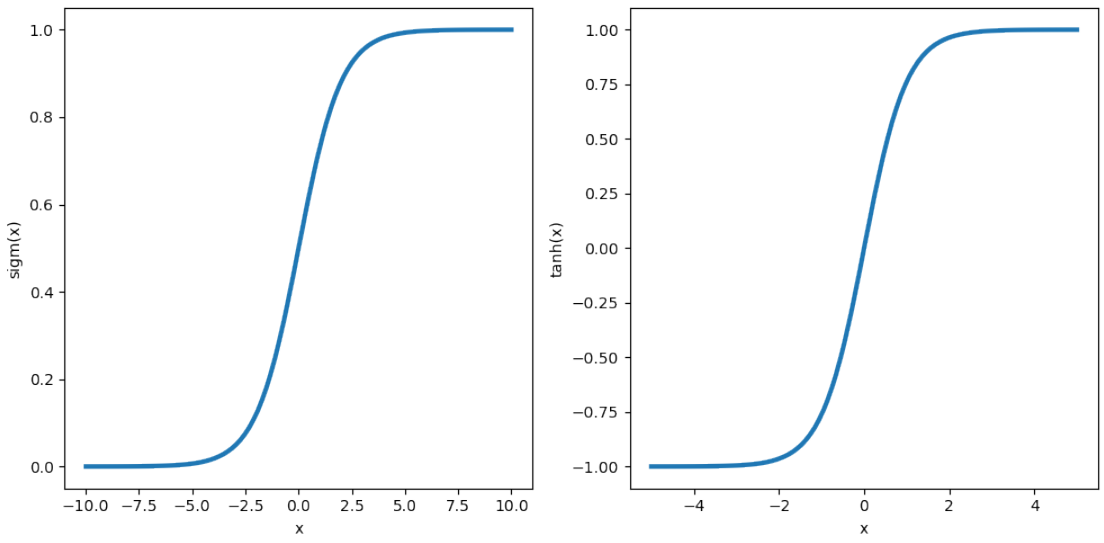

Figure 4: *The above plot shows on the left a sigmoid function and on the right a hyperbolicus tangent function. They look very similar but observe that the sigmoid has its asymptots at 0 and 1 while the tanh is at -1 and 1.*

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 $q(\mathbf{x})$

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 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.

2.4 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.

3 Summary

In this worksheet we have looked at how we can perform approximative inference of latent variables in our models when the marginal likelihood is intractable. In this case we looked at a model where the intractability comes from the computation of the model. Now both of the approaches we have looked at has their benefits, the sampling approach is quite easy to code up but it is hard to know how well we are doing, the variational Bayes' approach does provide a mean of comparison as we can look at the bound but we also know that we will never achieve the true solution. Approximate inference takes a lot of time and is a key part of the skillset for a machine learner, I would say personally that the research we do in my group, 10% of the time is spent coming up with models, the rest of the time is spent coming up with ways of approximating them. However, its beyond the scope of this course to look too much into this and this is why we just gave you a "whistle-stop" lecture and this optional work-sheet.

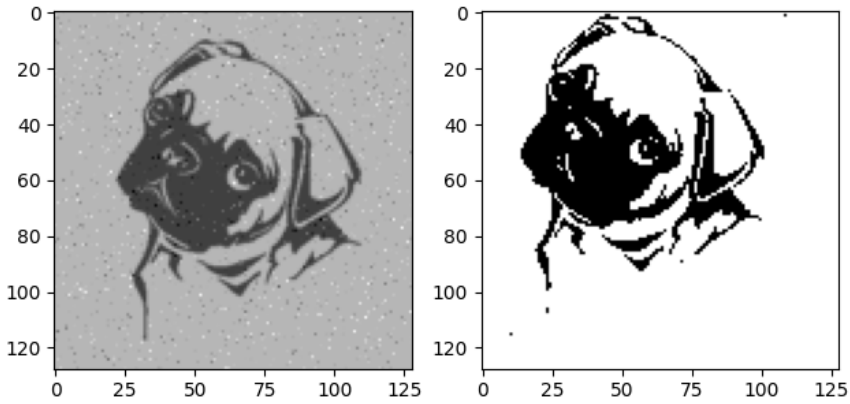

Figure 5: *This figure shows the result of my implementation of variational inference for the image denoising example. As you can see the ising prior cleans up the image rather nicely and we are left with a lovely black and white pug*