**1. Gaussian processes**

i)

A:

A long lengthscale results in underestimates of the variance

Long lengthscale assumes that there is high correlation between points in the function we are modelling, ie. the influence of a single data point spreads over a large distance

Priors with large lengthscale is too inflexible.

B:

As sigma -> 0 the variance decreases (overestimation) resulting in less uncertainty and smaller error bars around the function

As l -> 0 the exponent -> 0 again the variance decreases (overestimation) and we have smaller error bars, this is an effect of single points having a very low influence on nearby points

As l -> inf we have the opposite effect to before, the variance increases (underestimation) so we have larger error bars, single points have a large influence over nearby points

ii)

By adding independent kernels we end up with a resulting kernel that has a high value if either of the two summed kernels have a high value.

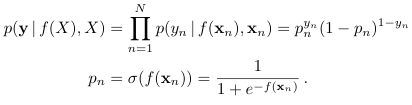
This means that we can model an underlying function that exhibits multiple characteristics that are not possible to model with just one of the kernels.

For example, if we add linear and periodic kernels then we get a model that is periodic with increasing mean away from the origin.

iii)

A:

Perform logistic regression by using Bernoulli likelihood to model the problem:



B:

There is no closed form solution for the marginal likelihood/posterior

C:

Estimate model params as a point rather than as a distribution and minimise negative log likelihood which can be evaluated with gradient descent methods

Alternatives:

Do a laplace approximation of the unnormalised posterior.

MAP solution (not as good answer I’d say though)

**b. Bayesian optimisation**

i)

A:

The acquisition function is used as a proxy for the objective function which is very expensive (or difficult) to compute.

So, we maximise the acquisition function instead of the objective function.

Alternative:

To choose where to evaluate the objective function next. Must balance the tradeoff between exploration and exploitation.

B:

In probability of improvement, we determine the probability that x\* leads to a better function value than the current best one we have found. This often leads to continued exploration around regions that the model is confident is good (low exploration - good exploitation).

In expected improis vement we instead quantify the amount of improvement that we expect if we take x\* next. This can lead to areas with large uncertainty being prioritised as they have high potential of being better (low exploitation - good exploration).

ii)

A:

Could apply BayesOpt to hyperparameter tuning for deep neural networks (expensive computation and black box). We would optimise an acquisition function given certain hyperparameters we want to tune and use the neural network as the objective function.

B:

Any task that involves an objective function that is easy to evaluate (say basic linear regression) so there is no use to approximating it with an acquisition function.

iii)

Maximising the marginal likelihood can go wrong early on in optimisation when we only have a few data points. We can solve this by integrating out the GP hyperparameters with MCMC sampling.

iv)

We could use neural networks, SVMs, random forests etc. as a surrogate model.

**2. Logisitic Regression**

a) i. Goes to infinity. As if linearly separable, if x > 0; class A, if x < 0 class B. So, any multiplier of x will fit perfectly, since its MLE (max), the max will be infinity.

a) ii. Not unique. Any large a is a maximum. b can vary, as long as decision boundary is between the linearly separable margin.

a goes to infinity (that was (a) i. right? whats this question)

b has multiple solutions (not clear which to pick)

b)

Decision boundary is when 0.5 = σ(θ0 + θ1x1 + θ2x2)

So when 0 = θ0 + θ1x1 + θ2x2

rewriting it, we have: x1 = - θ2x2 / θ1 - θ0/θ1  that’s in the form y = mx + c

So I guess that’s linear, not sure what we’re meant to see tbh

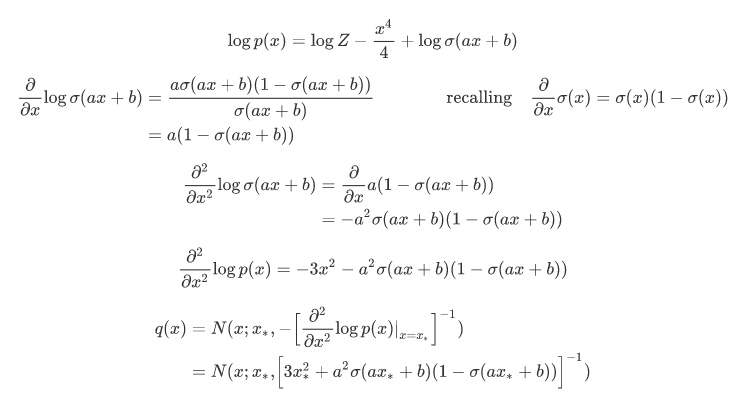
c) i. Gaussian

c) ii. Simple, makes calculation/analytic forms possible, may allow for good posterior approximation.

c) iii. requires integral (marginal likelihood) which is intractable (probably), no normalisation.

d) i. Approximate an (unormalised, if you like) distribution p, by a gaussian distribution, q.

d) ii.



level of detail required is not clear, 2nd derivative may not be required explicitly.

latex for above in case of edits:

\begin{align\*}

\log p(x)

&=

\log Z -\frac{x^4}{4} + \log \sigma(ax+b)

\end{align\*}

\begin{align\*}

\frac{\partial }{\partial x} \log \sigma(ax+b)

&=

\frac{a\sigma(ax+b)(1 - \sigma(ax+b))}{\sigma(ax+b)}

\qquad\qquad

\text{recalling} \quad \frac{\partial}{\partial x} \sigma(x) = \sigma(x)(1-\sigma(x))

\\

&=

a(1 - \sigma(ax+b))

\end{align\*}

\begin{align\*}

\frac{\partial ^{2}}{\partial x^{2}} \log \sigma(ax+b)

&=

\frac{\partial}{\partial x}

a(1 - \sigma(ax+b))

\\

&=

-a^{2} \sigma(ax+b)(1 - \sigma(ax+b))

\end{align\*}

\begin{align\*}

\frac{\partial ^{2}}{\partial x^{2}}\log p(x) = -3x^{2} -a^{2} \sigma(ax+b)(1 - \sigma(ax+b))

\end{align\*}

\begin{align\*}

q(x)

&= N(x; x\_{\*}, -\Big[ \frac{\partial^{2}}{\partial x^{2} } \log p(x) |\_{x=x\_{\*}} \Big]^{-1} )

\\

&= N(x; x\_{\*}, \Big[ 3x^{2}\_{\*} + a^{2} \sigma(ax\_{\*} +b)(1 - \sigma(ax\_{\*}+b)) \Big]^{-1} )

\end{align\*}

d) iii. Very not-gaussian. Not symmetric, heavy tailed, support is very different [0,1]

d) iv.

local (LA captures only local properties of the distribution)

huge (Bernstein-von Mises theorem)

not-unique (e.g. approximating uniform dist, there’s a lot of modes for that)

1. **Variational Inference**

a)

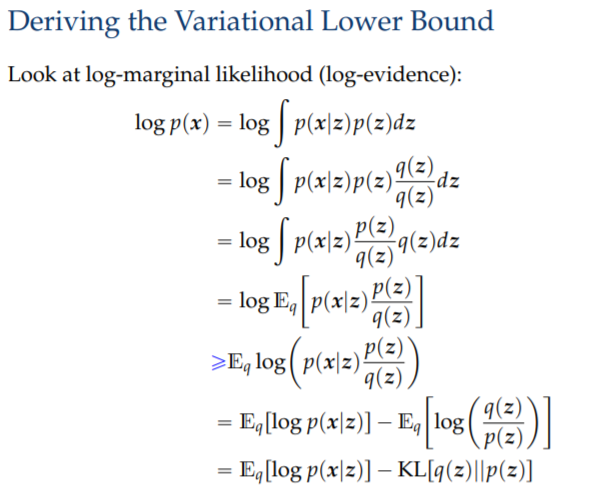
i) Aim of VI is to approximate intractable distribution p with tractable, parametrized distribution q.

ii)   
-start with log p(x) (why?)

= log (integral (p(x|z) p(z) dz )

-use Jensen inequality

-rearrange to ELBO



iii)

E\_q (log(p(x|z))) is data-fit term – the higher the better q(z) ‘predicts’ the x

KL[q(z)||p(Z)] is the regularizer – it prevents overfitting as it penalises the difference between the q(z) and the prior p(z)

iv)

KL[q(z) || p(z|x)] = E\_q(log q(z) - log p(z|x) ) = E\_q[log q(z) - log [p(x|z)p(z)/p(x)] ]

E\_q(log q(z) - [ log p(x|z) + log p(z) - log p(x)] ) =

E\_q(log p(x)) -[ E\_q(log p(x|z)) - E\_q(log q(z) - log p(z) ] =

log p(x) - [E\_q [log p(x|z)] - KL (q(z) || p(z) )] =

log p(x) - ELBO

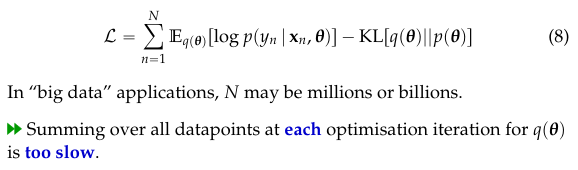
argmin\_q KL[q(z) || p(z|x)] <=> argmin\_q log p(x) - ELBO <=> argmax\_q ELBO

b)

i) in traditional VI each sample x\_i has corresponding latent variable z\_i, the distribution of which is learned through optimization. It is computationally expensive to calculate gradient w.r.t. all those variables if the dataset is huge.

alternative:

Huge datasets mean likelihood is product of huge number of terms => log likelihood is sum over huge number of terms. That’s too big and makes it too slow. See slide:



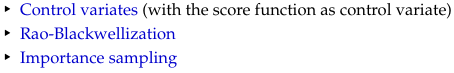
ii) With stochastic VI, instead of calculating the exact gradient, we use mini-batching to calculate stochastic gradient. I.e., we consider a random subset of samples x in each step of optimization. Use SGD.

iii) formula 3 is useful because even when it is not possible to compute it in closed form - since it is an expectation it can be approximated using Monte Carlo methods.

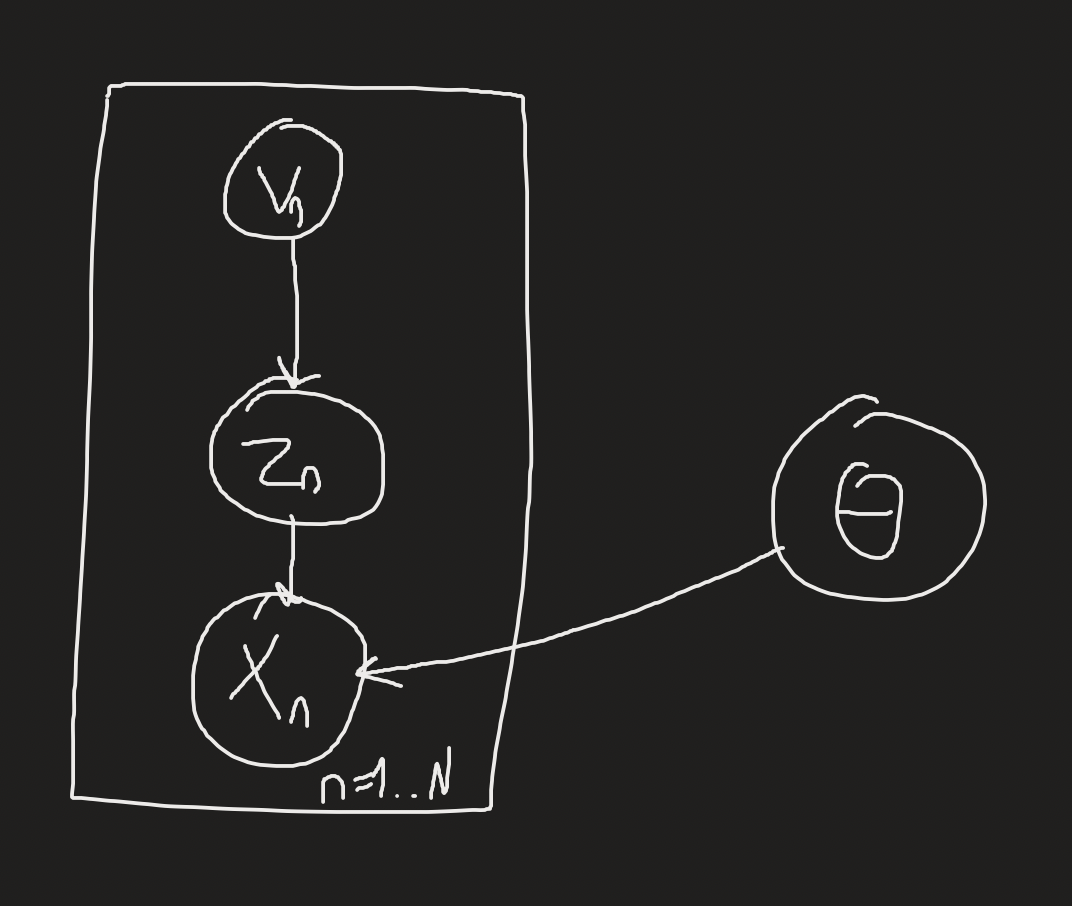
Also, this score function gradient is more widely applicable, such as to discrete random variables, where reparametrisation trick cannot be used. Allows for true *black-box* inference since it makes no assumptions about the problem.

iv)

Only the first is on syllabus (rest in stochastic VI slides which are reference only)



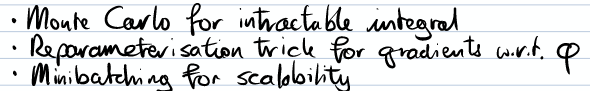
v)



vi)  
Variational Autoencoders use:

1.Reparameterization trick to write down the gradient w.r.t. v as an expectation

2.Monte Carlo. to approximate those gradients



**4. True or False?**

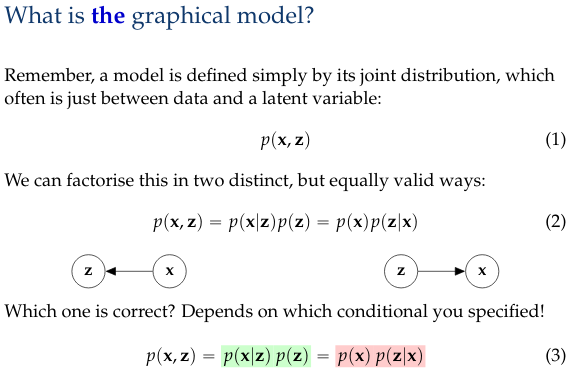
a)

i. True

ii. Not assessed

iii. True (maybe *any* is a bit of stretch, but that’s a bit subjective)

iv. Not sure this is assessed anymore? But maybe True, since the direction is actually arbitrary, slide:



b)

i. False (?? How do we train?)

ii. True (in general, I mean, if you choose you’re kernel well it could probably be convex)

iii. True

iv. True

c)

i. True

ii. False

iii. False

iv. True

d)

i. True? (maybe? this is kinda ambiguous, since it’s optimisation of an approximation...)

ii. False (although we’ve not covered these terms so not clear, but there’s no bayesian inference in VAE, so no prior)

iii. True

iv. False (should be convex)

e)

i. False

ii. True

iii. True, this is implied by ergodicity (although not equivalent)

iv. Unassessed