ProbAI Summer School Summary Notes

13/6/22 - 18/6/22

University of Helsinki

- Prior reading
- General github repo
- Lecture recordings

These notes are just a summary, all of the mathematical details and code are not duplicated here. There is a lot more detail in the lecture notes, accompanying notebooks and recordings.

General Introduction

- Mostly we care about how well the model works for *new inputs* drawn from the same distribution. This is easy if we have infinite data, but what if we have a limited sample?
- When we have finite data figuring out the right distribution is more difficult. This is where the Bayesian aspect comes in. We can learn a distrubtion over the parameter values conditional on whatever we have observed.
- Predictions are made by integrating over the uncertainty. Still have uncertainty even as the datasets get larger.
- Rare to be able to analytically calculate the posterior - so we have to approximate it!

How do we approximate the posterior?

- MCMC - common method but don't cover much of this in this summer school
- Distributional approximation - slightly older method
- Flexible approximation (i.e. Normalising flow) - what a lot of the summer school will focus on

Introduction to Probabilistic Models - 13/6/22 - Antonio Salmeron

Lecturer's notes

Examples of probabilistic models:

- Predictive medicine
- Self driving cars
- Image generation imagen
- Land use - using satelite images to monitor protected areas

These examples:

- Operate in environments where there's lots of data
- Data doesn't cover every possible scenario -> uncertainty
- Use probabilistic models
- Use inference algorithms to carry out prediction and structure analysis.

Probabilistic models offer:

- Principled quantification of uncertainty
- Natural way of dealing with missing data

What is Machine Learning? Book by Tom Mitchell, 1997

Easy example: Linear regression

Q: Is linear regression really ML? linear regression notebook

A: Yes, the more data we give it, the better the model - it is learning from experience.

The most basic example of linear regression is not probabilistic - but we can look at it from a Bayesian perspective.

> There are two types of uncertainty:
>
> - **Aleatoric** - due to randomness, i.e. the variability in the outcome of an experiment due to random effects.
>
> - **Epistemic** - due to lack of knowledge.

Assume we want to predict Y from X. We assume a joint distribution $p(x, y)$ - this is Epistemic uncertainty - it is unreducible. We can predict Y using $p(y|x) = \frac{p(x,y)}{p(x)}$. What is $p(y)$ for a specific x? This is aleatoric uncertainty - it is reducible.

Aleatoric uncertainty can be reduced by gathering more data, or increasing the number of features.

We want proabilitic models which can operate in high dimensional spaces and that support efficient inference and learning.

Probabilitic **graphical** models offer:

- *Structured* specification of high dimensional distributions in terms of low dimensional factors.
- *Efficient* inference and learning taking advantage of the structure.
- *Graphical* representation is interpretable by humans.

In a frequentist approach, you would assume that the parameter is a fixed value - it is NOT a random variable. In a Bayesian approach, all parameters are random variables. Information about parameters can be included prior to observing data.

Distribution in a Bayesian model The prior distribution $\pi(\theta)$ The joint distribution of X, y

$$\psi(X, y) = f(x|\theta)\pi(\theta)$$

The prior predictive distribution of X

$$m(x) = \inf \int_\theta f(x|\theta)\pi(\theta)d\theta$$

The posterior distribution of θ

$$\pi(\theta|x) = \frac{f(x|\theta)\pi(\theta)}{\int_\theta f(x|\theta)\pi(\theta)d\theta}$$

The predictive distribution given $ x= { x_1, . . . , x_n } $

$$f(x_{n+1}|x) = \int_\theta f(x_{n+1}|\theta, x)\pi(\theta|x)d\theta$$

> Two distributions are conjugate if the prior and posterior follow the same distributions; in this case the prior is called the **conjugate prior**.

The previous method is a **fully bayesian** approach. Sometimes you don't need to fully compute the posterior, it's enough to know the posterior up to a constant (i.e. you don't calculate the denominator in Bayes). In which case we can estimate θ by the MAP (maximum A posteriori), or by the MLE (maximum likelihood estimator).

Bayesian networks

A Bayesian network over random variables $X_1, ..., X_n$ consists of

- a directed acyclic graph (DAG)
- A set of local conditional distributions

there are three possible network structures in a DAG:

- serial connection
- diverging connection
- converging connection

From the DAG can figure out which variables are conditional/dependent on one another.

> **Generative vs. Discriminative models**
>
> *Generative* - learn $p(x, y)$ from data. Compute $p(y|x)$ using Baye's rule.
>
> - Examples: Naive Bayes, Bayesian networks in general, ...
> - Advantages: Can be used to generate synthetic data

  - Note: Higher asymptotic error but reached more quickly - so can be better for small datasets

Discriminative models - Estimate $p(y|x)$ directly from the data

  - Examples: Logistic rgression, NNs, ...
  - Note: Lower asymptotic error but reached more slowly

Graphical DAG notation:

- Plate notation - notation for drawing graphical DAGs that collapses the iterations.
- Variables in a grey circle are observed. Variables in a white circle are unobserved.
- Variables with a circle around them are modelled as random variables.

Using Monte Carlo can be a good first way to understand what your posterior is.

Example: factor analysis model

- In general, latent variable models are regarded as probabilistic models where some of the variables cannot be observed.
- Factor analysis summarises a high dimensional observation X of correlated variables by a smaller set of factors Z which are assumed independent a priori.

Inference in Bayesian networks

Have variables $X = (X_1, ..., X_n)$, and a bayesian network over them. And evidence X_E.

Inference Methods

Exact

- Brute force: compute $P(X, X_E)$ and maginalise out $X \setminus X_I$.

- Take advantage of the network structure, and do variable elimination.

Inference

- Sampling
- Deterministic

Monte Carlo inference algorithms

- If the entire population was available, the inference problem could be solved exactly, basically just count the cases.
- In reality we never have all the data.
- Monte Carlo operates by drawing an artificial sample from it using some random mechanism.
- The sample is used to estimate each variable of interest.

- Two main methods: Importance sampling and MCMC.

Importance sampling

- Fundamental idea is to express the computation we want to solve in terms of the expectation of the random variable.

- We sample for a different population, p^*. p^* is chosen to be very simple, it can be anything as long as it is non-negative for the same values as p. The accuracy depends on the choice of p^* - if we choose p^* as something close to p then we get better accuracy.

MCMC

- Have initial values for variables in the network.
- New item is generated conditional on previous configuration of variables.
- Elements of the sample are no longer independent - they form a Markov Chain.

- Danger is that you get stuck in one part of the population (local optima).

Probabalistic Modeling and Programming - Andres R. Masegosa, Thomas D. Nielsen

Scikit learn cheat sheet for choosing a model.

> PPL = Probabilistic programing language

Q: Why PPLs?

A:

- stacked architecture
- simplify probabilistic machine learning code
- reduce development time and cost to encourage experimentation
- reduce the necessary level of expertise
- democratisation of the development of probabilistic ML systems

---

Brief history of PPLs

90s/early 2000s - 1st generation of PPLs

- Turing complete probabilistic programming languages
- Used monte carlo methods
- Didn’t scale to big data, or higher dimensional data

2nd generation of PPLs

- Inference engine based on message passage algorithms
- Scaled to large datsets and high dimensional data

- Limited probabilistic model family

3rd generation of PPLs - where we are now

- pyTorch, Pyro, PyMC etc…
- Black box variational inference and hamiltonian monte carlo

Practise

See jupyter notebook here

> Pyro allows you to visualise the model, so that you can see dependencies `pyro.render_model(model)`

Example - In the ice-cream shop model:

This is coded up in the notebook.

- N - number of data points
- α, β are coefficients for the linear desciption of how ice-cream sales depend on temp. They are random variables.
- μ is the mean about which temperature is distributed. it is a random variable.
- t_i is the actaully temperature on a given day i, it is distributed normally about the mean μ.
- $s_{t,i}$ is the observed temperature as observed from a sensor. It is dependent on the temperature. It is distributed normally about the actual temperature with a std deviation of 1 to account for variation in the measurement.
- s_i is the number of sales on day i. It is a poisson distribution, depending on the temperature that day.

Bayesian Workflow - Elizaveta Semenova

- Code for examples on github.

General principles of bayesian inference: Specify a complete Bayesian model then sample the posterior distribution of the parameter θ.

PPLs are designed to let the user focus on modelling while inference happens automatically. Users need to specify the prior and likelihood.

Diagnosing MCMC outputs

Convergence diagnostics

- $$\hat{R}$$
- traceplots

Effective sample size (ESS)

- In MCMC samples will be autocorelated within a chain. This increases the uncertainty of the estimation of posterior quantitites.
- ESS is the number of independent samples to obtain the same level of uncertainty as from the available dependent samples.

We use multiple chains and inspect convergence after warm up (discard burn in period). Ideally we want all the chains to looks stationary and agree with one another. We can also look at the posterior distributions for each chain.

Modern Bayesian workflow

Modern Bayesian workflow is quite complicated - Paper by Gelman.

Prior predictive checking

Prior predictive checking consists in simulating data from the priors.

- Visualise priors (especially after transformation)
- This shows the range of data compatible with the model
- It helps understand the adequacy of the chosen priors

Iterative model building

A possible realisation of the Bayesian workflow loop:

- Understand the domain and problem (background knowledge)
- Formulate the model mathematically
- Implement model, test, debug
- Perform prior predcitive check
- Fit the model
- Assess convergence diagnostics
- Perform posterior predcitive check
- Improve the model iteratively, from baseline to complec and computationally efficient models.

Example: ODE-based compartmental model (SIR model)

Do a prior predictive check: calculate a range of possible trajectories for the ODE (so no use of observed data yet). Then we check whether our observed data lies in this range - if it does this indicates that the model is appropriate, as it's possible that the chosen priors and model give rise to the observed data.

Might find that our model isn't very good. Perhaps it doesn't pass our prior predictive check. Perhaps there is poor mixing of the chains. Think - what have we missed? If it was an SIR model of Covid maybe we haven't taken account of under-reporting, or the incubation period.

Example: Drug concentration response

- Concentration response experiments are used to rank drug candidates.

- Traditionally the drugs yield sigmoidal curves - characterised by a plateau at a high drug concentration.
- Curves show a loss of effect at higher doses, known as a 'hook effect'

Domain understanding - what do the clinical experts expect the model to show? Well looking to fit a curve that is:

- flat at low concentrations
- is able to capture the hook effect.

Can describe this using traditional Hill's model. Maybe this model isn't the best fit? Use a more flexible approach: Gaussian processes allow for a flexible curve shape.

We look at diagnostics, trace plots, chain and posteriors. Suppose when we look at the posterior it seems a bit over-fitted, what then?

Can use kernel design - this allows us to specify a wider range of gaussian process priors.

This is a new model and complicated. To better understand it we run prior predictives - this helps us to understand how the parameters change the model.

Conclusions

- use domain knowledge (expert guidance on the real-world problem)
- priors informed by domain knowledge
- using external data to point at a latent variable
- build to a more complex model

No one size fits all! Bayesian workflow is all about updating and considering what we're doing as we go.

Variational Inference and Optimization 1 - Helge Langseth, Andres Masegosa, Thomas Nielsen

Part 1 (Before lunch)

GitHub repo for this here.

Idea of variational inference: approximate $p(\theta|D)$ using $q(\theta|D)$. Note: as shorthand write $q(\theta) = q(\theta|D)$. Then do abunch of maths, so that finding $q(\theta)$ is an optimisation problem.

> Terminology
>
> - KL - Kullback Leibler divergence
> - ELBO - evidence lower bound
> - MF - mean field assumption
> - CAVI - coordinate ascent variational inference algorithm

Formalisation of approximate inference through optimisation. Given a family of tractable distributions Q and a distance measure Δ:

$$\hat{q} = argmin_{q \in Q} \Delta(q(\theta)||p(\theta|D))$$

.

Note: ideally Δ would be a metric (i.e. positive, symmetric and satisfy triangle inequality). However generally we use the Kullback-Leibler divergence (even though this isn't a proper metric).

Kullback-Leibler divergence

$$KL(f||g) = \int_\theta f(\theta) log(\frac{f(\theta)}{g(\theta)} = \mathbb{E}_{\theta \sim f}[log(\frac{f(\theta)}{g(\theta)})]$$

Variational Bayes setup

Variational bayes uses information projection. We want to approximate $p(\theta|D)$ by

$$\hat{q}(\theta) = argmin_{q \in Q} KL(q(\theta)||p(\theta|D))$$

We can rearrange KL as

$$KL(q(\theta)||p(\theta|D)) = logp(D) - L(q)$$

where L is the ELBO function.

ELBO

$$L(q) = E_q[log\frac{p(\theta, D)}{q(\theta)}]$$

Setup:

- we have observed our data D, and can calculate the full joint $p(\theta, D)$.
- we use the ELBO as our objective and assume $q(\theta)$ factorises.
- posit a variational family of distributions $q_j(|\lambda_j)$. So we choose the distributional form while optimising parameters λ_j.

Algorithm:

Repeat the following until neglible imporvements in terms of $L(q)$ * for each j: choose λ_j to maximise $L(q)$ based on D and $\{\lambda_i\}_{i \neq j}$ * calculate the new $L(q)$.

After some maths (given in the presentation/notes) we get:

$$L(q) = -KL(q_j(\theta_j)||f_j(\theta_j))) + c$$

The ELBO is maximised wrt q_j by choosing it equal to $f_j(\theta_j)$:

$$q_j(\theta_j) = \frac{1}{Z} exp(\mathbb{E}_{q\neg j}[logp(\theta, D)])$$

To get there we make two main assumptions:

- *Mean field*: assume $q(\theta) = \prod_i q_i(\theta_i)$, specifically $q(\theta) = q_j(\theta_j)q_{\neg j}(\theta_{\neg j})$.
- we optimise with respect to $q_j()$ and keep others fixed - i.e. do coordinate ascent.

The variational updating rules are guaranteed to never decrease the ELBO - i.e it should always monotonically increase. If it's not monotonically increasing then you know you've gone wrong!

Example: a simple Gaussian model

See the jupyter notebook here

Variational Inference and Optimization 2

Part 2 (After lunch :smile:)

GitHub repo for this here.

So far we have mostly considered linear models BUT a linear model might not be the right choice - so lets consider a neural network.

It's quite easy to fit a curve using a standard neural network - but we only get one solution. Instead, we want to use a Bayesian neural network.

Stochastic gradient descent

We want to optimise the ELBO using gradient descent.

> gradient ascent algorithm for maximising a function $f(\lambda)$
>
> - initialise $\lambda^{(0)}$
> - then update according to
>
> $$\lambda^{(t)} \leftarrow \lambda^(t-1) + \rho \cdot \nabla_\lambda f(\lambda^{(t-1)})$$

Standard gradient ascent is not enough for ELBO optimisation.

We won't be able to calculate the gradient exactly since:

- we may have to resort to mini-batching (calculating the gradient for a 'random subset')
- even for a mini batch may not be able to calculate the gradient exactly.

Black box variational inference (BBVI) Idea: cast inference as an optimisation problem.

Gradient ascent uses the gradient (derivative) to update the values. I.e. we take a sample, then update using the gradient ascent, and this will narrow into the desired solution. (It's sort of comparable to the sampling and updating step in MCMC).

Score function gradient

The notebook 'students_BBVI' gives a really useful example of this.

Reparameterised gradient

Can look at re-parameterising the distributions. This only works for cts distributions since $logp(\theta, D)$ needs to be differentiable wrt θ.

Interesting paper on this topic: Advances in variational research

Score function gradient vs. reparameterised gradient

Score function gradient:

- gradients point towards the mode of the approximation
- only requires $lnq(\theta|\lambda)$ to be differentiable.

Reparameterisation gradient:

- requires $q(\theta|\lambda)$ to be reparametrisable.
- requires $lnp(D, \theta)$ and $lnq(\theta|\lambda)$ be differentiable.

The reparametrisation trick is really good **IF** you have a function which can be reparameterised - not every function can be! The reparametrisation trick doesn't use the mean field approximation.

Summary of reprameterisation Reparameterisation: gradients align with model's gradient. But:

- requires $q(\theta|\lambda)$ to be reparametrisable.
- requires $lnp(D, \theta)$ and $lnq(\theta|\lambda)$ be differentiable - i.e. no categorical variables.

Automatic variational inference in PPLs

How does this actually work in PPLs? We looked at a very manual example, in reality much of this is already encoded in Pyro.

- *Manual*: define your data and the model
- *Manual/automatic*: define the variational distribution
- *Automatic*: optimise the ELBO

Example: temperature and sensor

Inference problem: Have measures of temperature taken with a sensor.

$$temp \sim N(15, 2)$$

$$sensor \sim N(temp, 1)$$

$$p(sensor = 18, temp)$$

Pyro models:

- random variables `pyro.samples`
- observations `pyro.samples` with the `obs` argument

Inference problem:

$$p(temp|sensor = 18)$$

Variational solution:

$$minKL(q(temp)||p(temp|sensor = 18))$$

Pyro guides:

- guides are abitrary stochastic functions.
- guides produce samples for those variables of the model which are not observed.

```
# the observations
obs = ('sensor': torch.tensor(18.0))

def model(obs):
    # this is an initialisation value for the temp
    temp = pyro.sample('temp', dist.Normal(15.0, 2.0))
    sensor = pyro.sample('sensor', dist.Normal(temp, 1.0), obs=obs['sensor'])

# the guide
def guide(obs):
    a = pyro.param('mean', torch.tensor(0.0))
    b = pyro.param('mean', torch.tensor(1.0), constraint=constraints.positive)
    temp = pyro.sampe('temp', dist.Normal(a,b))
```

Note: signatures (the 'names' given to variables) should be the same for the model and guide.

Example of variational inference using Pyro here

Exmple of using Bayesian linear regression using Pyro here

Example of using Bayesian logistic regression using Pyro here

Variational Inference and Optimization 3

Part 3 (After the coffee break :smile:)

GitHub repo for this here.

More details are found in the lectures notes and accompanying jupyter notebook.

Recap and conclusions

Variational inference

- Provides a distribution approximation to $p(\theta|D$

- Objective: $argmin_{\lambda} KL(q(\theta|\lambda)||p(\theta|D))$ is equivalent to \$$argmax_{\lambda} L(q(\theta|\lambda))$.
- Mean-field is a divide and conquer strategy for high dimensional posteriors
- Main caveat - $q(\theta|\lambda)$ tends to underestimate the uncertainty of $p(\theta|D)$

Coordinate ascent variational inference

- analytical expressions for some models
- CAVI is very efficient and stable if it can be used
- In principle requires manual derivation of updating equations - but can use toolboxes like Pyro

Gradient-based variational inference

- provides the tools for VI over arbitrary probabilistic models
- directly integrates with the tools for deep learning

Probabilistic programming languages

- PPLs fuel the 'build - compute - critique - repeat' cycle
- these offer ease and flexibility of modelling

What's next?

- The 'VI toolbox' is reaching maturity
- no longer a research area, more a pre-requisite for probabilistic AI

Deep Generative Models - Rianna van der Berg

Github repo for notes here

Variational auto-encoders

Variational inference a review for statisticians

Assume data was produced by a generative model: $p(z|x) = p(x|z) \cdot p$. Where x is our observed data and z is our latent variable which we want to find out.

Mean field variational inference is one of the simplest approaches. It makes the assumption that $q(z)$ can be written as a product of distributions over every latent variable.

In variational inference we're always interested in minimising $KL(q||p)$. The factorised version of the possible posterior assumes all variables are independent, so it will underestimate the relationship between variables.

Both $KL(p||q)$ and $KL(q||p)$ are specific instances of α divergence.

> **Auto-encoders** - copy input to output whilst going through a bottleneck. The code in the botleneck should be a compressed reprentation of the data x. Can use auto encoders as generative models - they learn representations of the data and generate new data.

Variational inference: optimise paramters of $q(z|x)$ for each x seperately.

Amortised variational inference: model paramters of $q(z|x)$ with a neural network $nn_\theta(x)$ for each x.

This paper - an introduction to variational autoencoders has some nice details/overview of the reparameterisation trick. The reparameterissation trick rewrites z as a deterministic function which makes it easier to sample.

The reparametrisation trick relies on you being able to write z as a determinisitc function g.

Can easily reparametrise for gaussian distribtuions and also for full-covariance gaussian distributions (i.e. where the variables influence one another)

Evidence lower bound might sometimes not be good enough. Then turn to *marginal likelihood estimation: importance sampling.*

Advances in VAEs

- Ladder VAEs. Have more layers of latent random variables - in practise this isn't always great - so change the order of the dependencies of the random variables.
- NVAE. Can be used to produce high resolution pictures using autoencoders.

Normalising flows

Normalising flows for variational inference:

If the true posterior is complicated and $q(\psi)$ is a simple gaussian, we can never get that close to the true posterior.

I want to transform q so that I end up with something closer to the true posterior. This helps to get correlated distributions - i.e. where variables are dependent. To do this can apply normalising flows.

Start by sampling z_0 from a simple distribution. Then going to apply a sequence of transformations to end up with soemthing more complicated. i.e. then apply a sequence of invertible transformations: $f_k : \mathbb{R}^D \to \mathbb{R}^D$.

$$z_k = f_k \prod f_{k-1} \prod ... \prod f_1(z_0)$$

and for each transformation $z_k = f_k(z_{k-1})$. End up with a way of getting $lnq_k(z_k)$ that allows for more complicated relationships between latent variables.

There are some practical requirements for normalising flows for variational inference:

- need flexible invertible transformations - as inflexible transformations lead to long sequences of flows for flexible posteriors. By flexible it means we want transformations which can do quite a lot.

-

  easily computable jacobian determinants, as we need to know this to calculate $lnq_k(z_k)$.

Q: What would these transformations look like in practise?

A: Planar flows. These is invertible, but not that flexible. However the jacobian determinant is easy to compute - which is where this method really shines.

Generative normalising flows

What about using these alone as a generative model (i.e. without autoencoders)? Associate the final transformed random variable to the observed data. Then will take the inverse of the transformations. This leads to an additional practical requirement - the inverse function needs to be easy to compute.

Denoising diffusion models

Idea: take data, assume it corresponds to a random variable. Apply a series of transformations to corrupt it into something with noise. Then reverse process - train a probabilistic model which matches the denoising at each step. I.e. at each step try and make it a little bit less noisy.

Normalising Flows - Didrik Nielsen

Lectures slides are here

Normalising flows describe the change of probably density through a series of invertible maps.

Q: Where are flows useful?

A: anywhere you need a flexible density $p(x)$.

Q: Why flows?

A: They can exactly get the density estimate, and it's quite fast. Sampling is also fast.

Normalising flows are useful for generative modelling and also for variational inference.

The framework

To construct $p(x)$ the key things you need are:

- Base distribution $p(z)$
- and the mapping f

we require f to be invertible/bijective so that we can map x to z, and z to x.

Can use it for sampling:

$$z \sim p(z)$$

$$x = f(z)$$

and can also use it for density:

$$logp(x) = logp(z) + log|det\frac{\partial z}{\partial x}|$$

$$z = f^{-1}(x)$$

The important parts are:

- *Forwards*: $x = f(z)$
- *Inverse*: $z = f^{-1}(x)$
- *The jacobian determinant*: $det\frac{\partial z}{\partial x}$

Deriving the change of variable formula:

- In one dimension: $p(x) = p(z)|\frac{dz}{dx}|$
- In higher dimensions: $p(x) = p(z)|det\frac{\partial z}{\partial x}|$

Need to structure the flow to make the jacobian easier to compute.

Can stack multiple bijections. The composition of bijections is a bijection.

Coupling flows

How to build efficient flows?

All about developing layers that:

- are expressive
- are invertible
- are cheap to compute their jacobian determinants.

Main categories of flows are:

- Det. identities
- Autoregressive
- Coupling - the most popular type of flow, fast in both directions and relatively easy to compute
- unbiased

Coupling flows

Forward:

$$x_{1:d} = z_{1:d}$$
$$x_{d+1:D} = \exp^{-\alpha_{d+1:D}} \cdot (z_{d+1:D} - \mu_{d+1:D})$$

Inverse:

$$z_{1:d} = x_{1:d}$$
$$z_{d+1:D} = x_{d+1:D} \cdot e^{\alpha_{d+1:D}} + \mu_{d+1:D}$$

Attempted to implement this in python, notebook for this is here

Autoregressive flows

Autoregressive flows are either fast for sampling or fast for density - but not both. If you decide to use autoregressive flows then need to use autoregressive NNs. Can use *masked autoregressive NNs*. For example: MADE (for vectors), WaveNet (for sequences) or PixelCNN (for images).

Continuous-time flows

Paper on neural ODE example.

$\frac{\partial z(t)}{\partial t} = f_\theta(z(t), t)$

Uses an instantaneous change of variables formula.

Related to score based diffusion models

Residual flows

$z = f(x) = x + h(x)$ for h - contractive function (lipshitz cts?).

Use hutchinson's trace estimator to calculate the jacobian.

In the paper they truncate the sum in hutchinsons trace estimaor. This introduces bias - which they get round this by swapping truncation for the russian roulette estimator.

Discrete flows

It's hard to transform a discrete distribution.

Discrete flows: $z_d = (\mu_d + \sigma_d x_d) mod K$

Integer Discrete flows: $z_d = x_d + \mu_d$

Both of these papers used a **straight through estimator**.

Surjective flows

surVAE Flows - authored by the lecturer.

Can consider flows which are no longer bijective but instead surjective.

Examples include:

- $x = round(z)$
- $x = z[:n]$ 'tensor slicing'
- $x = argmax(z)$

Bonus: Variational Inference with flows

Maximising the likelihood is exactly the same as minimising the KL.

Variational inference

Learn posterior approximation: $q_\lambda(\theta) \sim p(\theta|D)$

Mean field Gaussian: The common choice (which is traditional/'old-fashioned') is to use the mean field approximation, choosing each distribution to be Normal.

We can adapt this, so where we would use a mean field approximation we can instead use a flow.

Q: What kind of flows do we want to use for variational inference?

A: Well, desired property would be : fast sampling.

Gaussian Processes - Arno Solin

Pragmatic introduction to Gaussian processes

- A random vector is said to have a multivariate Gaussian distribution if all linear combinations of x are Gaussian distributed.
- A gaussian process can be considered as a distribution over functions $f : X \to \mathbb{R}$
$$f(x) \sim GP(\mu(x), \kappa(x, x'))$$
- A Gaussian process is completely defined by its mean $\mu(x)$ and covariance $\kappa(x, x')$.

Challenges/caveats of GPs

Three main challenges

1. *Scaling to large data.* A naive solution to deal with the expanded covariance matrix requires $O(n^3)$ compute.

2. *Dealing with non-conjugate likelihoods.*
3. *Representational power.* Gaussian processes are ideal for problems where it is easy to specify meaningful priors. If you can't… it becomes harder.

Challenge 1: Scaling to large data

The computational bottleneck can be tackled by:

- exploiting the structure of the data
- exploiting the structure of the GP prior
- solving the linear system approximately - i.e. conjugate gradient solvers
- splitting problem into smaller chunks. Split domains, chunk the data, …
- approximating the problem
- approximating the problem solution (SVGP - sparse variational gaussian processes).

Challenge 2: Dealing with non-conjugate likelihood models

- MCMC - accurate but generally heavy
- Laplace approximation - fast and simple - but old fashioned!
- Expectation approximation - efficient but tricky, requires lots of tuning
- Variational methods - dominant method today

Challenge 3: Representational power

- GPs can be seen as shallow but infinitely wide. This might not be the right model for the job!!! i.e would be a bad choice for a low dimensional manifold in a high dimensional space.
- BUT, they can be good tools to combine with other models!

Connections and approaches to GPs

Connection to Neural Networks

- showed that untrained single layer NNs converge to GPs in the limit of infinite width.

Connection to physics

- models often written in terms of ODEs/PDEs etc
- GPs used as structured priors.

Connection to Bayesian optimisation

- used to figure out the next optimal point to look at - seems interetsing, could this be used for healthcare decision modelling?

Recap

- Gaussian processes provide a plug and play framework for probabilistic inference and learning
- give an explicit way of understanding prior information in a problem

- provide meaningful uncertainty estimates and means for quantifying uncertainty.

Good (old-ish) book - Gaussian processes for machine learning - Rasmussen.

Neural ODEs - Cagatay Yildiz

This follows the jupyter notebook 'NODE.ipynb' here

Simulation-based inference - Henri Pesonen

Use simulators for real world phenomena. The complexity of the simulators prohibits access to likelihood function.

Example: Transmissions of bacterial infections in daycare centers

- Cross sectional data from SIS model
- Continuous time model.
- Want to simulate which child in which daycare centre carries which strain of data?

Example: Personalised medicine

- Model how cancer cells are evolving in the tissue.
- Want to simulate how the combination of drugs is kiling the cancerous cells.

Simulator

A computer program defined as $x \approx p(x|\theta)$ that has

- input paramters θ
- stochastic output x

The data can be in any format, i.e. single time series, images, distributions of data points

Inference

Observe the data and infer the values of the parameters that generated them. This could use likelihood, or a bayesian approach, or MLE.

Inference without likelihood

- use the capability to draw simulated data conditioned on the input parameters.
- based on assumption that likely true parameters values produce data that is similar enough to the observed data.

Distance metric

Distance metric used to measure how close the simulated data is to the observed data.

- Acceptable region defined by distance metric
- Choice of metric depends on the data format, e.g. could use Eulcidean or L1 etc

Data dimensionality

- In high dimensional spaces it becomes hadrer to generate data close to the observed data.
- Current approach is to use summary statistics $S(\cdot)$.

$$d(x, x^o) \sim d(S(x), S(x^o))$$

- Sufficient statsictics usually don't exist.

How do we select summary statistics?

- open problem
- use bespoke summary statistics - could use *domain expertise*, explore the simulator prior to inference, diagnose the inference results.
- also automatic algorithms for selecting/constructing summary statistics.

Putting this together get: **Rejection Approximate Bayesian Computation (ABC)**

for $i = 1, ..., N$

$$\theta^* \sim p(\theta)$$

$$x^* \sim p(x|\theta^*)$$

while

$$d(x^*, x^0) > \epsilon$$

then

$$\theta^* \sim p(\theta)$$

$$x^* \sim p(x|\theta^*)$$

set $\theta_i = \theta^*$

Alternative approach

Instead of choosing a fixed threshold can sample a large artifical set and choose a fraction of samples which are most similar to the observed data. Threshold can then be calculated based on which samples were selected.

Rejection ABC uses samples from the prior.

Unless we have lots of prior information about the parameters, sampling from the prior isn't effective. So can instead sequentially formulate importance sampling distributions with more probability mass in interesting regions - Sequential Monte Carlo ABC (SMC-ABC).

Issues to be aware of

Summary statistics may not catch relevant features of the data

- can decrease dimension too much thus losing information
- didn't decrease dimension enough, so didn't solve the problem
- can be correlated - have redundant dimensions

Multiple summary statistics can have widely different scales, so unequal contributions to the distance metric.

Book on this subject: Handbook of approximate bayesian computation

Surrogate models

Alternative apporach is to construct surrogate models of parts of the system.

Example: construct approximate likelihood at an abritrary parameter value.

Synthetic likelihood is hardly efficient - the surrogate is fitted at each parameter value seperately.

Bayesian optimisation for likelihood free inference (BOLFI)

This approach uses Gaussian processes.

Gaussian process surrogates can utilise active learning.

- Different strategies for selecting parameter values where to query the simulator. This reduces the number of queries to produce reasonable approximations to posterior ditribution.

Want to find parameter values that minimise the discrepancy function. Uses black box optimisation. Aquisition strategies balance exploration and exploitation.

Minimising the distance may not be optimal. Want to choose query points that are most informative about the model.

How to minimise the distance?

Aquisition functions

- *Lower Confidence Bound Selection Criterion* Lower confidence bound selection criteria for minimising the distance.
- *MaxVar* the maximum variance aquisition method
- *RandMaxVar* - randomised version. Randomised Maximum Variance
- *ExpIntVar* - Most efficient method: the expected integrated variance. The drawback of this method is that it is quite difficult to calculate.

Sampling from surrogate

- to represent the posterior distribution we require a sample drawn from it. Can use MCMC

After inference

How reliable are the results?

Different error sources:

- algorithm performance
- model performance
- simulator performance

Likelihood free inference methods are based on several levels of approximations - these all add to the total error.

Notebook for tutorial using ELFI (lecturer's python package for doing simulation inference) is here

The generative model is described as a DAG.

When doing BOLFI, doing NUTS may not be the best choice. You only need to choose the next point to evaluate at, so may as well just use metropolis hastings.

NUTS sampler is default for the posterior sampler phase.

Human-centered ML - Fani Deligianni

Computing technologies for healthcare team

Why do we need human-centered AI?

- acountability
- technical robustness and safety
- oversight
- privacy and data governance
- non discrimination and fairness
- transparency
- soietal and enviromental wellbeing

Healthcare models - how do we go from ML models to something which can be used? Paper here

ABCD Guide

- A: callibration in the large (this means external validation), or the model intercept
- B: calibration slope
- C: discrimination with the reciever operating characteristic curve
- D: clinical usefulness with decision curve analysis

Is risk under or over estimated? Look at predicted risk vs. observed proportion.

Think about clinical consequences, what matters more sensitivity or specificity?

i.e. in breast cancer a false-negative is more harmful than a false-positive.

Would need to do decision analysis - explicit validation of health outcomes. Can look at net benefit.

Net benefit

$netbenefit = sensitivity * prevalence - (1 - specificity) * (1 - prevalence) * \frac{thresholdprobability}{1 - thresholdprobability}$

Compare the model with treating all positive, treating none or treating based on another factor, i.e. duration of illness.

Experts can inform the threshold for treating people.

Uncertainty

- Frequentist approach provides confidence intervals.

What guarantees can we use against dicriminatory bias?

- Look at calibration within groups.

Transparency and explainability

- explainability is required to ensure impartial decision making process
- people have a right to know why decisions were made for them.

Think about who is the target audience of explainability

- clinicians - need to be able to trust the model
- patients - need to understand decisions
- data scientists/developers
- investors

Traditionally use interpretable models in healthcare situations. But maybe we should make more of an effort to explain more complicated models. i.e. NN can be explainable models.

How do we explain a model?

- Think about local (personal decision) based explanations vs. more global explanations
- For example, when talking to a patient how can we explian the model in a way which is relevant to their experience?

Paper on: The next generation of medical decision support

Bayesian Neural Networks - Bayesian Neural Networks 101 - Yingzhen Li

Lecture slides are here. And lecturers notes are here.

Bayesian solution:

- Put a prior $p(\theta)$ on network parameters θ, e.g. gaussian prior
- approximate bayesian predictive inference:
- monte carlo approximation

Steps for approximate inference in BNNs

1. construct the $q(\theta) \approx p(\theta|D)$ distribution.
2. Fit the $q(\theta)$ distribtuion. e.g. using variational inference.
3. Compute prediction with Monte Carlo predictions

Part 1: Basics

the true posterior: $p(\theta|D) = p(D|\theta)p(\theta)/p(D)$

have an approximate posterior: $q(\theta)$

Use Kullback-Leibler Divergence (KL). (note: when p=q, KL=0, otherwise KL>0.)

We want to minimise $KL[q(\theta)||p(\theta|D)] = logp(D) - E_q[log\frac{p(\theta,D)}{q(\theta)}$

So minimising KL, is eqivalent to mkaximising $L = E_q[log\frac{p(\theta,D)}{q(\theta)}$. This is the veidence lower bound (ELBO).

Can re-write the ELBO: $L = E_{q_\psi(\theta)}logp(\theta, D) - KL[q_\psi(\theta)||p(\theta)]$.

Using other q distributions

Can use more complicated q distributions.

- Pro: more fleixble approximations -> better posterior approximations
- Con: higher time and space complexities

Can use *last-layer BNN*.

- use deterministic layers for all but the last layer
- for the last layer use full covariance gaussian approximate posterior.
- For regression this is equivakent to bayesian linear regression (BLR) with NN-based non-linear features. We use a KL regulariser for the last layer only.

Can make it more economic using *MC-dropout*.

- add drop out layers to the network
- perform drop out during training

Case study 1: Bayesian optimisation (BO)

first idea: fit a surrogate function, $f_\theta \approx f_0$.

So have $y_i \approx f_0(x_i) + \epsilon$

Idea of BO: iterate the following steps - fit a surrogate function f_θ with uncertainty estimates. - use a surrogate function to guide the dataset collection process.

Start from a samll amount of observations and update the dataset using $D = D \cup \{(x_*, y_*)\}$.

Can use the upper confidence bound (UCB) as an acquisition function.

Case study 2: Detecting adversarial examples

Hypothesis:

- adversarial examples

Uncertainty measures

total uncertainty = epsitemic uncertainty + aleatoric uncertainty

imagine flipping a coin

epistemic uncertainty how much do I believe the coin is fair?

aleatoric uncertainty what's the next coin flip outcome?

Can use shannon entropy to compute uncertainty. $H[p] = -\sum plog(p)$

Applications

BBNs in medical imaging - Super resolution

Linearised laplace approximation in Bayesian Neural Networks - Miguel Hernandez Lobato

Slides are here.