Bayesian Neural Networks for Text Classification and Regression

Wilker Aziz
ILLC @ UvA

Let's predict hate speech

Remember this exercise from week 2b?

Data We have a dataset of observations of the kind (x, y) where x is a relatively short piece of text and y is a binary flag that indicates the presence of hate speech.
Model $Y|\theta, x \sim \text{Bern}(f(x; \theta))$ where f is an NN
Problem It looks like sometimes the classifier makes mistakes. We are asked to provide the **model's uncertainty about its predictions**.

Options

1. We can report the probability $p(y|x, \theta)$ which can be easily obtained from our model. We are happy to have chosen a probabilistic approach!
2. Our model does not support a meaningful notion of uncertainty, it predicts probability distributions **deterministically** and therefore the probabilities themselves have no variance. We need another module from DL4NLP!

This may shock you, but $p(y|x, \theta)$ is not an uncertainty estimate.

Here is the source of confusion. A decision y, given x and θ, is not what the model predicts. The model predicts a probability value $p(y|x, \theta)$, or more generally, **a probability distribution** $Y|x, \theta$. This prediction is itself deterministic and therefore has no variance. In a strong sense, $f(x; \theta)$ is *absolutely certain* about the probability value it's predicted.

Intuition Would you take Donald Trump's word on his involvement in corruption? Would your answer change should Donald Trump say "Oh, I am 1000% sure I am not involved in corruption. And by the way, I bet you've never seen a probability so large, it is, it is so large, it's huge, seriously, best probability ever".

Uncertainty

A parameter, along with NN architectures and every assumption we make, **singles out a function** $f(x;\theta)$.

Deeper down it also specifies another function, namely, the probability mass/density function $p(y|x,\theta)$ that assigns likelihood to observations.

Perturbing the parameter, even just a little, may reveal that this function changes dramatically for certain x.

But why should we care about a perturbation in parameter space? After all, haven't we chosen our parameters for a reason?

For example, $|f(x;\theta+\epsilon) - f(x;\theta)|$ may be very large!

Some parameter estimation algorithm gives us θ, and it is pretty clear all throughout training that many different instances of θ can do just as well (under our criterion, say likelihood). So small perturbations in parameter around our solution should not be so frowned upon.

I'd actually stretch and say perturbations should not be frowned upon, no matter how tiny or large (something like Euclidean distance in parameter space is quite irrelevant here).

Uncertainty

But I want to make a strong argument. So, let's choose perturbed parameters **very** carefully.

Say we have a stochastic process that generates instances of θ that are *supported by our data* $\mathcal{D}$. This hypothetical process denoted $\Theta|\mathcal{D}$ sees data and narrows our parameter options to good parameters only.

Imagine a random variable $Q = p(y_*|x_*, \theta)$ for some novel (x_*, y_*), where θ is a sample from the hypothetical process $\Theta|\mathcal{D}$.

If $\mathrm{Var}(Q)$ is large, the model knows very little about the probability of y_* given x_*. Think of it this way, our **very best** instances of θ, *those which are very likely given* $\mathcal{D}$, fail at assigning a reasonably consistent probability value to the data point.

How can we obtain such a *conditional* view $\Theta|\mathcal{D}$?

Maybe x_* is in a region of $\mathcal{X}$ that's barely represented in $\mathcal{D}$.

Uncertainty

A model's assessment of its own uncertainty is not the likelihood value $p(y|x, \theta)$, nor the related quantity $\mathrm{Var}(Y|x, \theta)$.

Instead, it's the distribution of that likelihood value for instances of θ that are likely given all data we have already observed.

Uncertainty is represented by $Y_*|x_*, \mathcal{D}$ and can be quantified, for example, by $\mathrm{Var}(Y_*|x_*, \mathcal{D})$.

Variance of $Y|x, \mathcal{D}$ is about *uncertainty*, variance of $Y|x, \theta$ is just about *likelihood*.

For a moment, let me use $\mathcal{M}$ to explicitly denote all of our model assumptions (i.e., conditional independencies, parametric families, NN architectures). Then

- $\mathrm{Var}(Y|X = x, \mathcal{D}, \mathcal{M})$ is an assessment of the uncertainty of a model $\mathcal{M}$ about Y given x taking into account all hypotheses supported by a set of observations $\mathcal{D}$;
- $\mathrm{Var}(Y|X = x, \Theta = \theta, \mathcal{M})$ is the variance of Y given x under a specific hypothesis θ compatible with model $\mathcal{M}$;
  The data $\mathcal{D}$ plays no role here (except perhaps indirectly via an algorithm for picking θ).

Uncertainty is represented/captured by the distribution of a random variable. Uncertainty about unobserved rvs thus requires a Bayesian view of the world. Luckily for us, Bayesian theory requires very little more than axiomatic probability theory (Bernardo and Smith, 2009). Well, and a bit of creativity to approximate some tough computations.

Outline

1. Bayes: what and why?
2. Choosing a prior
3. Posterior Inference for BNNs
4. Bayesian Dropout
5. Example

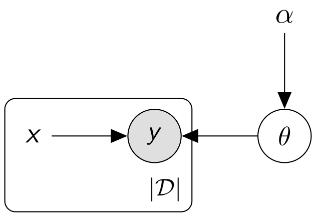
Bayes: what and why?

In this section we aim to answer

- What is a Bayesian neural net (BNN)?
- Why should we care about them?

What's a BNN?

BNN



Regression

$$Y|\theta, x \sim \mathcal{N}(\mu(x;\theta), \sigma(x;\theta)^2)$$

Classification

$$Y|\theta, x \sim \mathrm{Cat}(\pi(x;\theta))$$

with for example $\theta \sim \mathcal{N}(\underbrace{0, I}_{\alpha})$

- as before, NNs power the mapping from x and θ to $p(y|x,\theta)$
- though now θ is a *random variable* distributed according to a prior $p(\theta|\alpha)$

Just a joint distribution!

Observations are rvs, and *parameters* are rvs.

In a Bayesian model, data and parameters are no different. They are all given random treatment. The only substantial difference, being observed or not, has no effect in the theory. Read it this way: the set of principles is the same (axiomatic probability theory), there is no need for context-dependent patches.

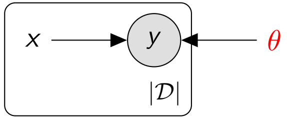
In a Bayesian model, the prior parameter α, sometimes called a hyper-parameter, is typically fixed (or itself governed by a distribution, whose parameter is fixed (or, itself governed by a distribution, whose parameter is fixed (or ...))).

A BNN is a Bayesian model with NN-parameterised likelihood.

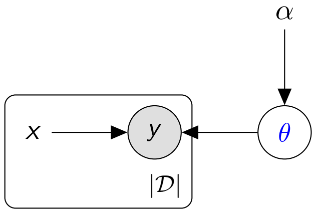
NNs and BNNs side by side

Probabilistic models powered by

NN



BNN



- NN: assumes θ to be **given**
- BNN: all variables are treated alike, that is, they are random variables whether or not we call them *parameters*

BNNs also have deterministic parameters (e.g α)
we call those *hyperparameters* and they are ideally *fixed*

Let's recap

- we have a **likelihood** $p(x|\theta)$, with which we can assign probability $p(\mathcal{D}|\theta)$ to data $\mathcal{D}$
- we have a **prior** $p(\theta|\alpha)$, it restricts the 'possible worlds' to some worlds that are plausible *a priori* (that is, before we look at this particular data $\mathcal{D}$)
- together they induce a **joint distribution** $p(\mathcal{D}, \theta|\alpha)$
- whose marginal $p(\mathcal{D}|\alpha) = \int p(\theta|\alpha)p(\mathcal{D}|\theta)\mathrm{d}\theta$ we also call **evidence**

NNs: parameters θ are known and given, which means, we need to find them somewhere. This view is so widespread that is common to think that optimisation and learning are the same thing.

BNNs: parameters θ are random and sampled from a prior. There's no search, there's no need for searching. Every single query of interest takes nothing but probability calculus. Here learning dispenses with optimisation.

Some Bayesians do optimise hyperparameters α, say using maximum (marginal) likelihood estimation $\arg\max_{\alpha} \ \log p(\mathcal{D}|\theta)$. Those Bayesians are known as *Empirical Bayesians*. There's something funny about this term, it makes it look like being Bayesian precludes empirical considerations. There's nothing *un-empirical* about Bayes.

Bayes

Being *Bayesian* seems to require specifying a prior distribution over parameters, though it's more than that

- it's about acknowledging that most quantities are *unknown*
- and proceeding to reason probabilistically under uncertainty
- priors are a means to this end, they specify what kinds of values are reasonable and with what expectation
- as we will see, acknowledging uncertainty and treating it seriously requires probabilistic inference

Why am I emphasising this? You will find arguments of the kind *"this regularisation is equivalent to that prior"*. The connection between a regulariser and a prior does not confer any Bayesian-type credibility to a non-Bayesian algorithm. These claims are usually made in the context of parameter estimation, they mostly only hold asymptotically (access to infinite data) and at global optima. Are these assumptions reasonable enough to justify some weak connection to Bayes? What's the purpose of the connection anyway? For example, remarking a connection for it inspires changes to the algorithm could be a good reason. Attempting to impose a perception of principledness is far less useful. In ML we have to compromise here and there *all* the time, there should be no shame in that. Still, motivating our compromises matters, it informs our peers, and we should make careful use of superficially powerful claims, after all, our goals include communicating research clearly.

Why do we call it a theory? Isn't it just a tool? A type of probability calculus? The motivations for the Bayesian paradigm are rooted in *a theory of rational decision making under uncertainty*, and in that sense it does go beyond probability calculus: it adds a semantic layer to it with philosophical implications (Bernardo and Smith, 2009). If you want to concentrate on statistical and practical data analysis implications, a textbook like BDA3 (Gelman et al., 2013) is more appropriate.

Why Bayes?

A fairly practical reason for Bayes stems from a question such as? How can we quantify the model's uncertainty about a *prediction*?

Wouldn't it be useful to shed light onto

- when do we know we can trust the model for a given pair (x, y)?

When I say *prediction*, does anyone still think of the following?

$$y_* = \arg\max_y \; p(y|x_*, \theta)$$

If so, let's agree on some terminology. This is a decision rule (it is not even the only one possible) and it relies on the likelihood $p(y|x_*, \theta)$

- think of the likelihood as a prediction on its own right
- an NN parameterised by θ has predicted this value from x_*

When Bayes? This is not like choosing your favourite cake, you *can* be objective about this. If Bayesian computations posed no challenging, I'd feel more like telling you *always Bayes*, at least, whenever your data are outcomes of random experiments. But that's not reality. So, learn about Bayes and decide when it's worth the trouble. That applies to all of our tools, doesn't it? Sometimes an NN is not worth the trouble: the time you save not acquiring expert knowledge about the problem goes to waste in silly numerical instability and fighting overfitting with extremely limited theoretical guidance.

The importance of knowing what we don't know

If $p(y_*|x_*, \theta)$ **is not** about uncertainty, then what is?

When a data point is well supported by θ, that is, $p(y_*|x_*, \theta)$ is high, we should ask ourselves, is θ even supported by the evidence we have?

In other words, we are interested in the posterior distribution $\Theta|\mathcal{D}, \alpha$:

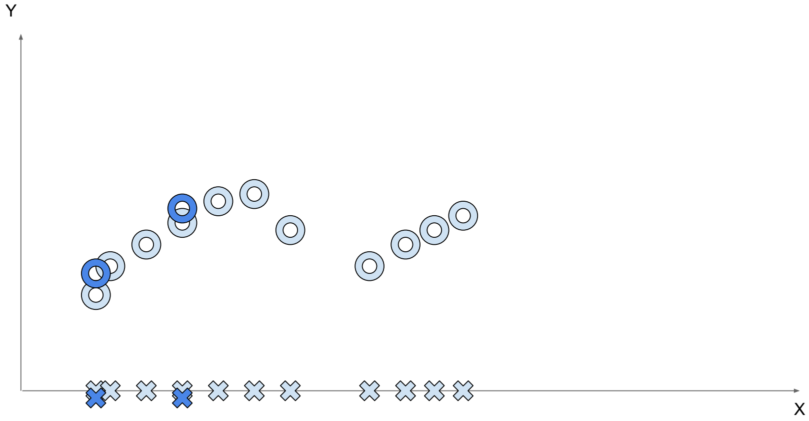
$$p(\theta|\mathcal{D}, \alpha) = \frac{p(\theta, \mathcal{D}|\alpha)}{p(\mathcal{D}|\alpha)}$$

To be able to get to it, we need to accept that θ is random, acknowledge that we don't know much more about it than what can be coded in a prior $p(\theta|\alpha)$, and proceed to reassess our beliefs in light of data $\mathcal{D}$.

"The importance of knowing what we don't know" (Gal, 2016, Chapter 1).

The loss landscape of NNs are full of large valleys: different parameters incur similar (possibly very little) loss, but give rise to different functions (Garipov et al., 2018). Though these functions perform well, they make meaningfully different predictions on test data. How do they differ away from training data? Can we even attempt to answer this questions if we have a single function?

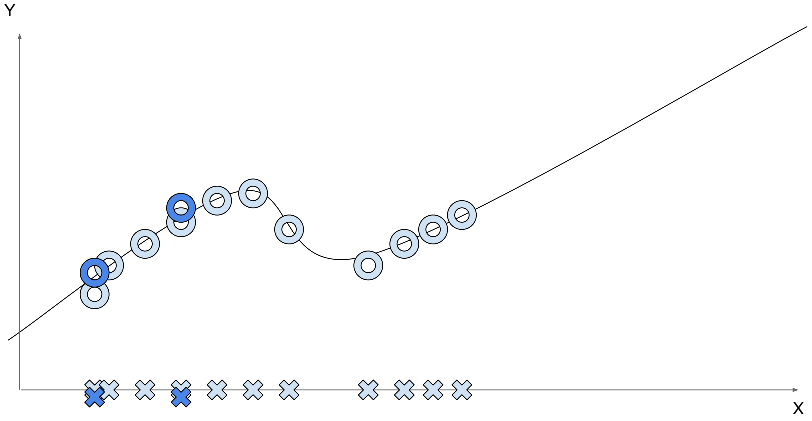
Uncertainty illustrated



Suppose a regression problem for which we have observations

Here we have data points for some regression problem which we could use to fit an NN (let's say we start with MSE).

Uncertainty illustrated

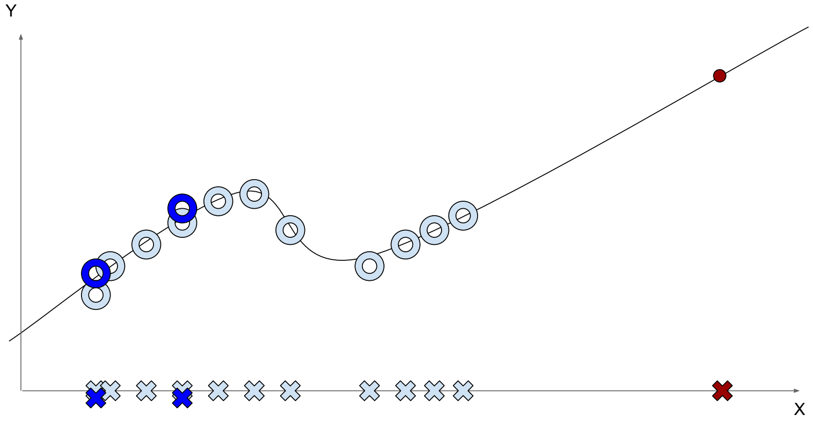


Let's approach with the help of NNs, i.e. $y = \text{NN}(x; \theta)$

Here we have data points for some regression problem which we could use to fit an NN (let's say we start with MSE).

- NNs are deterministic and cannot deal with observed variance. See the darker crosses overlapping the lighter crosses? Those are identical inputs with different responses. Best an NN can do is to predict the average response.

Uncertainty illustrated

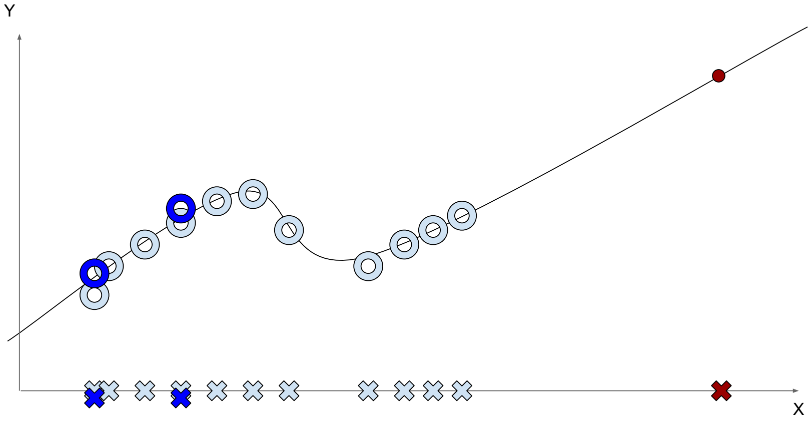


By design, it extrapolates predictions to unseen inputs, e.g. x_*

Here we have data points for some regression problem which we could use to fit an NN (let's say we start with MSE).

- NNs are deterministic and cannot deal with observed variance. See the darker crosses overlapping the lighter crosses? Those are identical inputs with different responses. Best an NN can do is to predict the average response.
- We use NNs (or ML in general) exactly because we hope to generalise some meaningful pattern to unseen inputs. One may query the model about an input that is far removed from the observed data. What should we do there?

Uncertainty illustrated

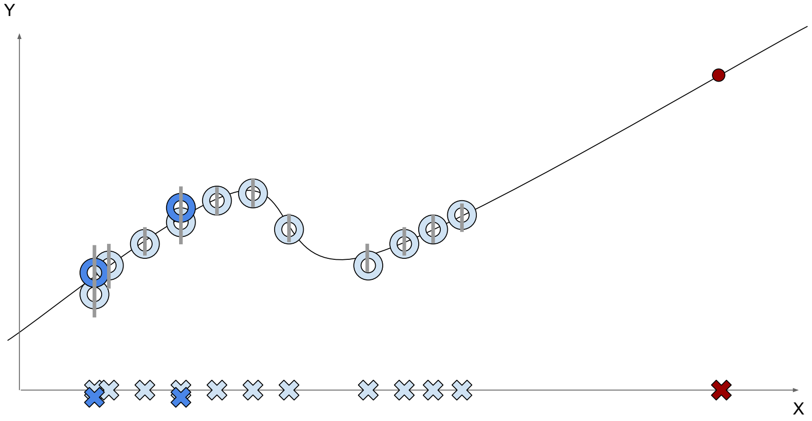


Can we trust our model given x_* is far from observations?

Here we have data points for some regression problem which we could use to fit an NN (let's say we start with MSE).

- NNs are deterministic and cannot deal with observed variance. See the darker crosses overlapping the lighter crosses? Those are identical inputs with different responses. Best an NN can do is to predict the average response.
- We use NNs (or ML in general) exactly because we hope to generalise some meaningful pattern to unseen inputs. One may query the model about an input that is far removed from the observed data. What should we do there?
- It looks like we need a mechanism to assess how much the mode knows about the input.

Uncertainty illustrated

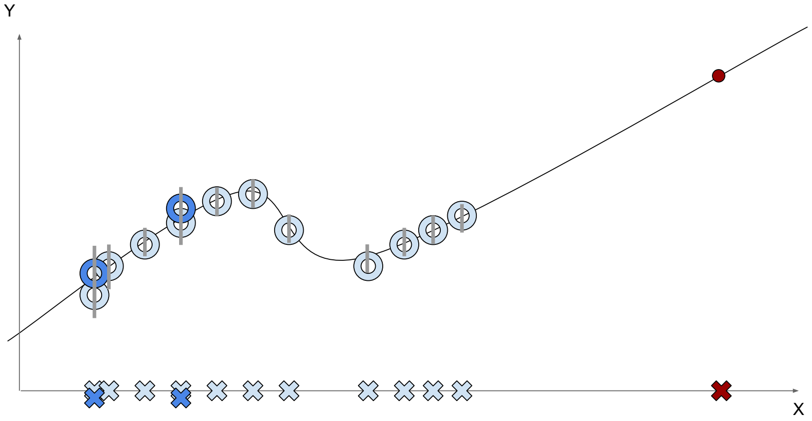


Let's fit Gaussians, i.e. $\mathcal{N}(\mu(x;\theta), \sigma(x;\theta)^2)$, around targets

Here we have data points for some regression problem which we could use to fit an NN (let's say we start with MSE).

- NNs are deterministic and cannot deal with observed variance. See the darker crosses overlapping the lighter crosses? Those are identical inputs with different responses. Best an NN can do is to predict the average response.
- We use NNs (or ML in general) exactly because we hope to generalise some meaningful pattern to unseen inputs. One may query the model about an input that is far removed from the observed data. What should we do there?
- It looks like we need a mechanism to assess how much the mode knows about the input.
- We know how to fit likelihood-based models, so let's model the data as conditionally Gaussian and learn to predict both Gaussian parameters.

Uncertainty illustrated

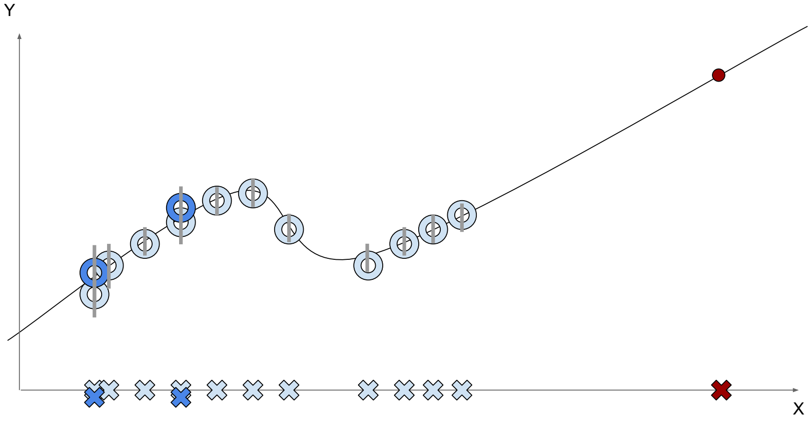


Note that we never observe much variability for a given input x

Here we have data points for some regression problem which we could use to fit an NN (let's say we start with MSE).

- NNs are deterministic and cannot deal with observed variance. See the darker crosses overlapping the lighter crosses? Those are identical inputs with different responses. Best an NN can do is to predict the average response.
- We use NNs (or ML in general) exactly because we hope to generalise some meaningful pattern to unseen inputs. One may query the model about an input that is far removed from the observed data. What should we do there?
- It looks like we need a mechanism to assess how much the mode knows about the input.
- We know how to fit likelihood-based models, so let's model the data as conditionally Gaussian and learn to predict both Gaussian parameters.
- It's hard to expect a model will learn to predict a pattern it does not observe. And look at this, we never observe much variance.

Uncertainty illustrated

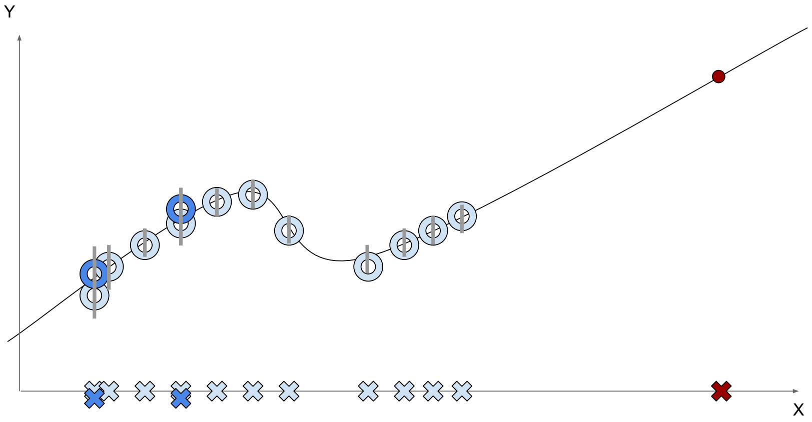


$\mu(x;\theta)$ learns to be on average close to every response for x

Here we have data points for some regression problem which we could use to fit an NN (let's say we start with MSE).

- NNs are deterministic and cannot deal with observed variance. See the darker crosses overlapping the lighter crosses? Those are identical inputs with different responses. Best an NN can do is to predict the average response.
- We use NNs (or ML in general) exactly because we hope to generalise some meaningful pattern to unseen inputs. One may query the model about an input that is far removed from the observed data. What should we do there?
- It looks like we need a mechanism to assess how much the mode knows about the input.
- We know how to fit likelihood-based models, so let's model the data as conditionally Gaussian and learn to predict both Gaussian parameters.
- It's hard to expect a model will learn to predict a pattern it does not observe. And look at this, we never observe much variance.

Uncertainty illustrated

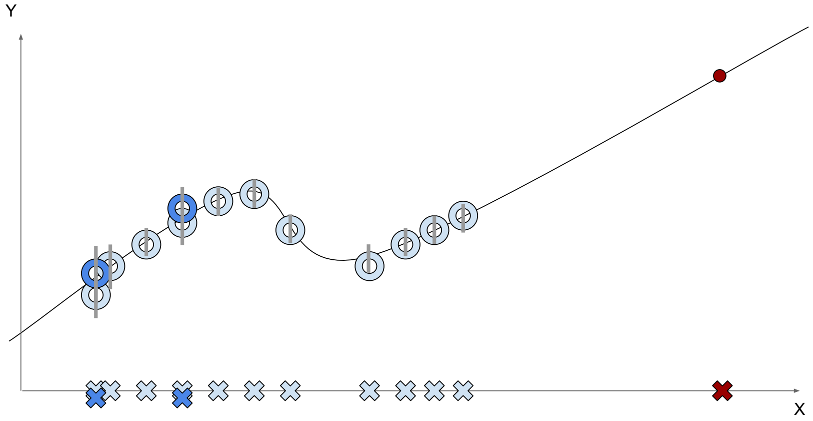


$\sigma(x;\theta)$ instead learns to cover all responses for x, but no more

Here we have data points for some regression problem which we could use to fit an NN (let's say we start with MSE).

- NNs are deterministic and cannot deal with observed variance. See the darker crosses overlapping the lighter crosses? Those are identical inputs with different responses. Best an NN can do is to predict the average response.
- We use NNs (or ML in general) exactly because we hope to generalise some meaningful pattern to unseen inputs. One may query the model about an input that is far removed from the observed data. What should we do there?
- It looks like we need a mechanism to assess how much the mode knows about the input.
- We know how to fit likelihood-based models, so let's model the data as conditionally Gaussian and learn to predict both Gaussian parameters.
- It's hard to expect a model will learn to predict a pattern it does not observe. And look at this, we never observe much variance.

Uncertainty illustrated

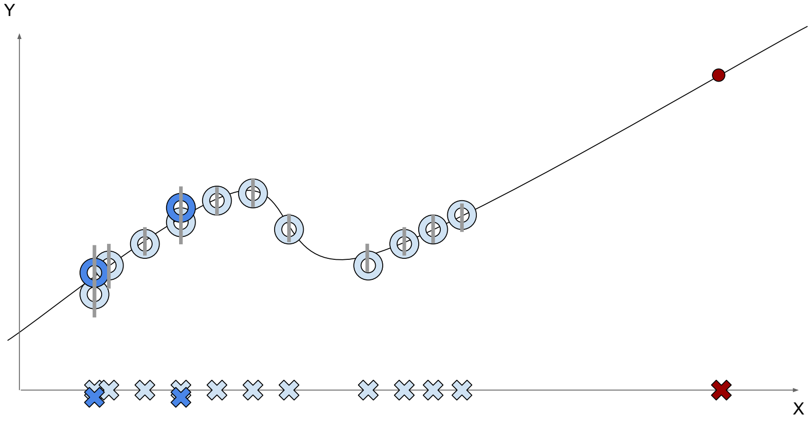


for MLE does not like covering more than observed responses

Here we have data points for some regression problem which we could use to fit an NN (let's say we start with MSE).

- NNs are deterministic and cannot deal with observed variance. See the darker crosses overlapping the lighter crosses? Those are identical inputs with different responses. Best an NN can do is to predict the average response.
- We use NNs (or ML in general) exactly because we hope to generalise some meaningful pattern to unseen inputs. One may query the model about an input that is far removed from the observed data. What should we do there?
- It looks like we need a mechanism to assess how much the mode knows about the input.
- We know how to fit likelihood-based models, so let's model the data as conditionally Gaussian and learn to predict both Gaussian parameters.
- It's hard to expect a model will learn to predict a pattern it does not observe. And look at this, we never observe much variance.
- The Gaussian model looks better than MSE, but only where we have data.

Uncertainty illustrated

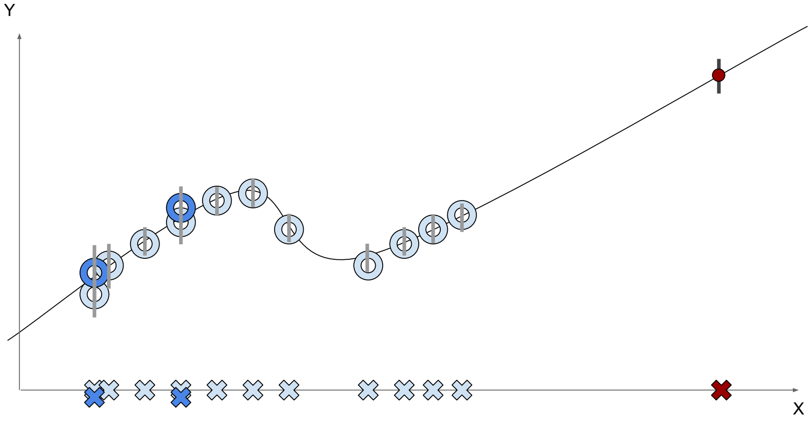


What is our expectation for $\sigma(x_*; \theta)$?

Here we have data points for some regression problem which we could use to fit an NN (let's say we start with MSE).

- NNs are deterministic and cannot deal with observed variance. See the darker crosses overlapping the lighter crosses? Those are identical inputs with different responses. Best an NN can do is to predict the average response.
- We use NNs (or ML in general) exactly because we hope to generalise some meaningful pattern to unseen inputs. One may query the model about an input that is far removed from the observed data. What should we do there?
- It looks like we need a mechanism to assess how much the mode knows about the input.
- We know how to fit likelihood-based models, so let's model the data as conditionally Gaussian and learn to predict both Gaussian parameters.
- It's hard to expect a model will learn to predict a pattern it does not observe. And look at this, we never observe much variance.
- The Gaussian model looks better than MSE, but only where we have data.

Uncertainty illustrated

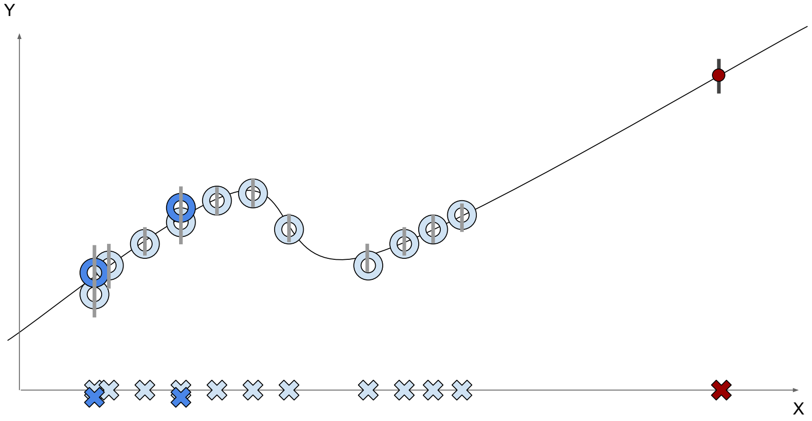


For all we know, $\sigma(\cdot;\theta)$ likes to predict small values

Here we have data points for some regression problem which we could use to fit an NN (let's say we start with MSE).

- NNs are deterministic and cannot deal with observed variance. See the darker crosses overlapping the lighter crosses? Those are identical inputs with different responses. Best an NN can do is to predict the average response.
- We use NNs (or ML in general) exactly because we hope to generalise some meaningful pattern to unseen inputs. One may query the model about an input that is far removed from the observed data. What should we do there?
- It looks like we need a mechanism to assess how much the mode knows about the input.
- We know how to fit likelihood-based models, so let's model the data as conditionally Gaussian and learn to predict both Gaussian parameters.
- It's hard to expect a model will learn to predict a pattern it does not observe. And look at this, we never observe much variance.
- The Gaussian model looks better than MSE, but only where we have data.

Uncertainty illustrated

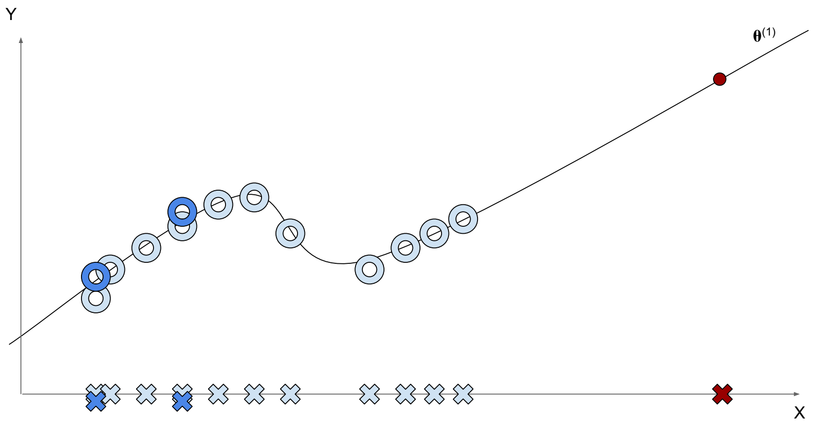


$\sigma(\cdot;\theta)$ seriously underestimates uncertainty for x_*

Here we have data points for some regression problem which we could use to fit an NN (let's say we start with MSE).

- NNs are deterministic and cannot deal with observed variance. See the darker crosses overlapping the lighter crosses? Those are identical inputs with different responses. Best an NN can do is to predict the average response.
- We use NNs (or ML in general) exactly because we hope to generalise some meaningful pattern to unseen inputs. One may query the model about an input that is far removed from the observed data. What should we do there?
- It looks like we need a mechanism to assess how much the mode knows about the input.
- We know how to fit likelihood-based models, so let's model the data as conditionally Gaussian and learn to predict both Gaussian parameters.
- It's hard to expect a model will learn to predict a pattern it does not observe. And look at this, we never observe much variance.
- The Gaussian model looks better than MSE, but only where we have data.

Uncertainty illustrated

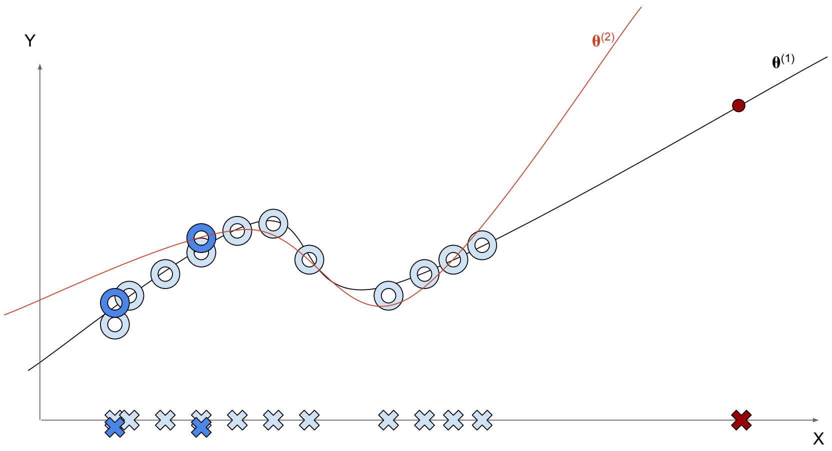


but what if, with probability $p(\theta^{(1)}|\mathcal{D})$, we consulted $\mu(x;\theta^{(1)})$?

Here we have data points for some regression problem which we could use to fit an NN (let's say we start with MSE).

- NNs are deterministic and cannot deal with observed variance. See the darker crosses overlapping the lighter crosses? Those are identical inputs with different responses. Best an NN can do is to predict the average response.
- We use NNs (or ML in general) exactly because we hope to generalise some meaningful pattern to unseen inputs. One may query the model about an input that is far removed from the observed data. What should we do there?
- It looks like we need a mechanism to assess how much the mode knows about the input.
- We know how to fit likelihood-based models, so let's model the data as conditionally Gaussian and learn to predict both Gaussian parameters.
- It's hard to expect a model will learn to predict a pattern it does not observe. And look at this, we never observe much variance.
- The Gaussian model looks better than MSE, but only where we have data.
- Ditch the idea of prediction variance. Instead, we impose a prior on θ and infer a posterior distribution given all of our observed data. Now let's consult patterns that are likely **given** data.

Uncertainty illustrated

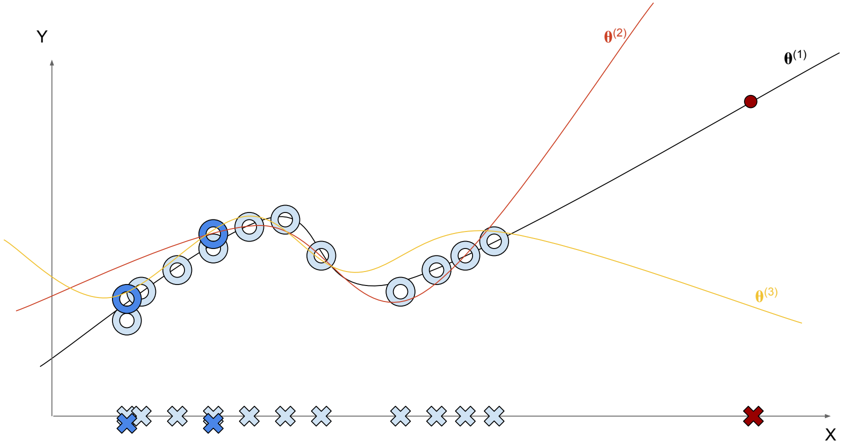


and $\mu(x;\theta^{(2)})$, with probability $p(\theta^{(2)}|\mathcal{D})$

Here we have data points for some regression problem which we could use to fit an NN (let's say we start with MSE).

- NNs are deterministic and cannot deal with observed variance. See the darker crosses overlapping the lighter crosses? Those are identical inputs with different responses. Best an NN can do is to predict the average response.
- We use NNs (or ML in general) exactly because we hope to generalise some meaningful pattern to unseen inputs. One may query the model about an input that is far removed from the observed data. What should we do there?
- It looks like we need a mechanism to assess how much the mode knows about the input.
- We know how to fit likelihood-based models, so let's model the data as conditionally Gaussian and learn to predict both Gaussian parameters.
- It's hard to expect a model will learn to predict a pattern it does not observe. And look at this, we never observe much variance.
- The Gaussian model looks better than MSE, but only where we have data.
- Ditch the idea of prediction variance. Instead, we impose a prior on θ and infer a posterior distribution given all of our observed data. Now let's consult patterns that are likely **given** data.
- You may be thinking 'hold on if I sample some NN parameters, I don't just get lucky and approximate observed responses well', but recall, you are sampling from $\Theta|\mathcal{D},\alpha$, these are the curves that are likely **given** data.

Uncertainty illustrated

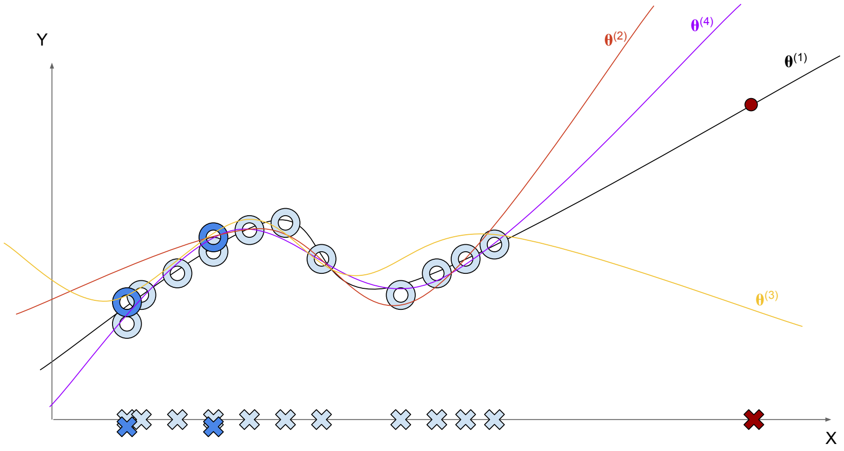


and $\mu(x; \theta^{(3)})$, with probability $p(\theta^{(3)}|\mathcal{D})$

Here we have data points for some regression problem which we could use to fit an NN (let's say we start with MSE).

- NNs are deterministic and cannot deal with observed variance. See the darker crosses overlapping the lighter crosses? Those are identical inputs with different responses. Best an NN can do is to predict the average response.
- We use NNs (or ML in general) exactly because we hope to generalise some meaningful pattern to unseen inputs. One may query the model about an input that is far removed from the observed data. What should we do there?
- It looks like we need a mechanism to assess how much the mode knows about the input.
- We know how to fit likelihood-based models, so let's model the data as conditionally Gaussian and learn to predict both Gaussian parameters.
- It's hard to expect a model will learn to predict a pattern it does not observe. And look at this, we never observe much variance.
- The Gaussian model looks better than MSE, but only where we have data.
- Ditch the idea of prediction variance. Instead, we impose a prior on θ and infer a posterior distribution given all of our observed data. Now let's consult patterns that are likely **given** data.
- You may be thinking 'hold on if I sample some NN parameters, I don't just get lucky and approximate observed responses well', but recall, you are sampling from $\Theta|\mathcal{D}, \alpha$, these are the curves that are likely **given** data.

Uncertainty illustrated

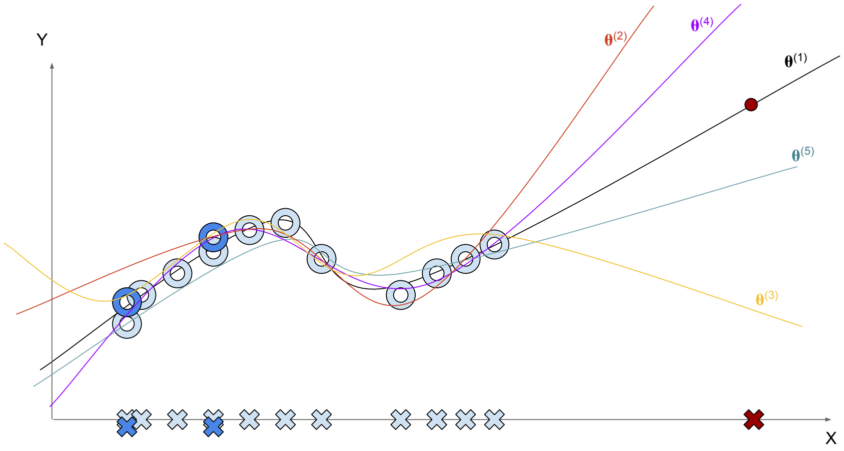


and $\mu(x;\theta^{(4)})$, with probability $p(\theta^{(4)}|\mathcal{D})$

Here we have data points for some regression problem which we could use to fit an NN (let's say we start with MSE).

- NNs are deterministic and cannot deal with observed variance. See the darker crosses overlapping the lighter crosses? Those are identical inputs with different responses. Best an NN can do is to predict the average response.
- We use NNs (or ML in general) exactly because we hope to generalise some meaningful pattern to unseen inputs. One may query the model about an input that is far removed from the observed data. What should we do there?
- It looks like we need a mechanism to assess how much the mode knows about the input.
- We know how to fit likelihood-based models, so let's model the data as conditionally Gaussian and learn to predict both Gaussian parameters.
- It's hard to expect a model will learn to predict a pattern it does not observe. And look at this, we never observe much variance.
- The Gaussian model looks better than MSE, but only where we have data.
- Ditch the idea of prediction variance. Instead, we impose a prior on θ and infer a posterior distribution given all of our observed data. Now let's consult patterns that are likely **given** data.
- You may be thinking 'hold on if I sample some NN parameters, I don't just get lucky and approximate observed responses well', but recall, you are sampling from $\Theta|\mathcal{D},\alpha$, these are the curves that are likely **given** data.

Uncertainty illustrated

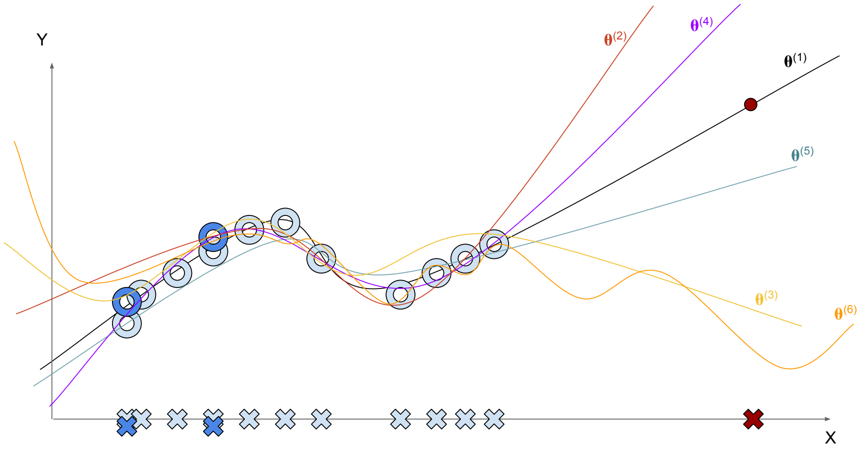


and $\mu(x; \theta^{(5)})$, with probability $p(\theta^{(5)}|\mathcal{D})$

Here we have data points for some regression problem which we could use to fit an NN (let's say we start with MSE).

- NNs are deterministic and cannot deal with observed variance. See the darker crosses overlapping the lighter crosses? Those are identical inputs with different responses. Best an NN can do is to predict the average response.
- We use NNs (or ML in general) exactly because we hope to generalise some meaningful pattern to unseen inputs. One may query the model about an input that is far removed from the observed data. What should we do there?
- It looks like we need a mechanism to assess how much the mode knows about the input.
- We know how to fit likelihood-based models, so let's model the data as conditionally Gaussian and learn to predict both Gaussian parameters.
- It's hard to expect a model will learn to predict a pattern it does not observe. And look at this, we never observe much variance.
- The Gaussian model looks better than MSE, but only where we have data.
- Ditch the idea of prediction variance. Instead, we impose a prior on θ and infer a posterior distribution given all of our observed data. Now let's consult patterns that are likely **given** data.
- You may be thinking 'hold on if I sample some NN parameters, I don't just get lucky and approximate observed responses well', but recall, you are sampling from $\Theta|\mathcal{D}, \alpha$, these are the curves that are likely **given** data.

Uncertainty illustrated

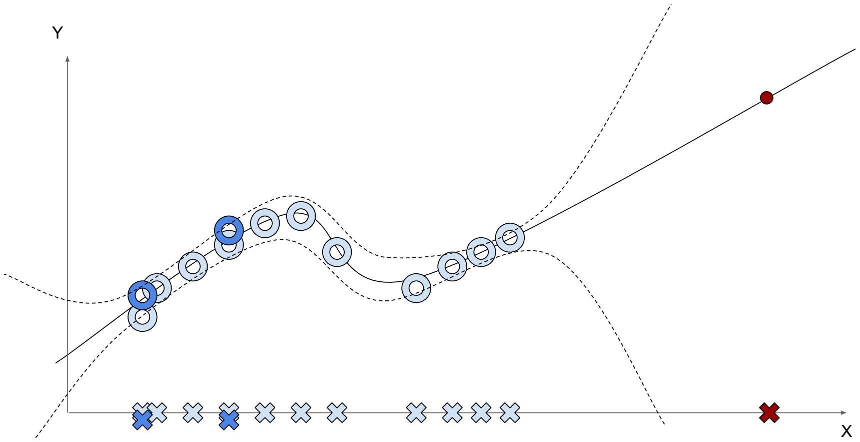


and $\mu(x;\theta^{(6)})$, with probability $p(\theta^{(6)}|\mathcal{D})$?

Here we have data points for some regression problem which we could use to fit an NN (let's say we start with MSE).

- NNs are deterministic and cannot deal with observed variance. See the darker crosses overlapping the lighter crosses? Those are identical inputs with different responses. Best an NN can do is to predict the average response.
- We use NNs (or ML in general) exactly because we hope to generalise some meaningful pattern to unseen inputs. One may query the model about an input that is far removed from the observed data. What should we do there?
- It looks like we need a mechanism to assess how much the mode knows about the input.
- We know how to fit likelihood-based models, so let's model the data as conditionally Gaussian and learn to predict both Gaussian parameters.
- It's hard to expect a model will learn to predict a pattern it does not observe. And look at this, we never observe much variance.
- The Gaussian model looks better than MSE, but only where we have data.
- Ditch the idea of prediction variance. Instead, we impose a prior on θ and infer a posterior distribution given all of our observed data. Now let's consult patterns that are likely **given** data.
- You may be thinking 'hold on if I sample some NN parameters, I don't just get lucky and approximate observed responses well', but recall, you are sampling from $\Theta|\mathcal{D}, \alpha$, these are the curves that are likely **given** data.
- This is showing a much more realistic picture about x_*, isn't it?

Uncertainty illustrated



Suddenly, we are a lot less certain about predictions for x_*

Here we have data points for some regression problem which we could use to fit an NN (let's say we start with MSE).

- NNs are deterministic and cannot deal with observed variance. See the darker crosses overlapping the lighter crosses? Those are identical inputs with different responses. Best an NN can do is to predict the average response.
- We use NNs (or ML in general) exactly because we hope to generalise some meaningful pattern to unseen inputs. One may query the model about an input that is far removed from the observed data. What should we do there?
- It looks like we need a mechanism to assess how much the mode knows about the input.
- We know how to fit likelihood-based models, so let's model the data as conditionally Gaussian and learn to predict both Gaussian parameters.
- It's hard to expect a model will learn to predict a pattern it does not observe. And look at this, we never observe much variance.
- The Gaussian model looks better than MSE, but only where we have data.
- Ditch the idea of prediction variance. Instead, we impose a prior on θ and infer a posterior distribution given all of our observed data. Now let's consult patterns that are likely **given** data.
- You may be thinking 'hold on if I sample some NN parameters, I don't just get lucky and approximate observed responses well', but recall, you are sampling from $\Theta|\mathcal{D}, \alpha$, these are the curves that are likely **given** data.
- This is showing a much more realistic picture about x_*, isn't it?

Oh... This is a Fancy Ensemble?

I want to say *yes*, because if we are going to talk about ‘ensembles’, then I feel like I owe a word such as ‘fancy’ to Bayes.

But honestly, I don’t feel like using the word ‘fancy’, because marginal and conditional probabilities are pretty basic.

Finally, I’m not sure it’s fair to explain something self-consistent (i.e., probability calculus) in terms of such a vaguely specified notion (i.e., ensembling). Perhaps we should explain ensembling as an attempt at probability calculus?

But let’s build intuition, and expose differences!

Oh... This is a Fancy Ensemble?

Not really, ensembles reduce stochasticity of prediction to data-independent initial conditions.

I want to say *yes*, because if we are going to talk about 'ensembles', then I feel like I owe a word such as 'fancy' to Bayes.

But honestly, I don't feel like using the word 'fancy', because marginal and conditional probabilities are pretty basic.

Finally, I'm not sure it's fair to explain something self-consistent (i.e., probability calculus) in terms of such a vaguely specified notion (i.e., ensembling). Perhaps we should explain ensembling as an attempt at probability calculus?

But let's build intuition, and expose differences!

Oh... This is a Fancy Ensemble?

Not really, ensembles reduce stochasticity of prediction to data-independent initial conditions.

For example, an ensemble might combine K independently converged maximum likelihood estimates (each time using a different random initalisation $\theta_0^{(i)}$):

- some will ensemble parameters of the likelihood:
  $\mu(x_*) = \frac{1}{K}\sum_{i=1}^{K} \mu(x_*; \theta^{(i)})$ for $\theta^{(i)} = \text{MLE}(\mathcal{D}, \theta_0^{(i)})$
  this can disregard any spread the MLE solutions might have captured
- others will ensemble likelihood assessments:
  $\frac{1}{K}\sum_{i=1}^{K} p(y_*|x_*, \theta^{(i)})$ for $\theta^{(i)} = \text{MLE}(\mathcal{D}, \theta_0^{(i)})$
  this requires a prediction a specific query y_*

I want to say *yes*, because if we are going to talk about 'ensembles', then I feel like I owe a word such as 'fancy' to Bayes.

But honestly, I don't feel like using the word 'fancy', because marginal and conditional probabilities are pretty basic.

Finally, I'm not sure it's fair to explain something self-consistent (i.e., probability calculus) in terms of such a vaguely specified notion (i.e., ensembling). Perhaps we should explain ensembling as an attempt at probability calculus?

But let's build intuition, and expose differences!

- Contrast this with stochasticity due to assessments of $\mu(x_*; \theta)$ or $p(y_*|x_*, \theta)$ for $\theta \sim \Theta|\mathcal{D}, \alpha$

Oh... This is a Fancy Ensemble?

Not really, ensembles reduce stochasticity of prediction to data-independent initial conditions.

For example, an ensemble might combine K independently converged maximum likelihood estimates (each time using a different random initalisation $\theta_0^{(i)}$):

- some will ensemble parameters of the likelihood:
  $\mu(x_*) = \frac{1}{K}\sum_{i=1}^{K} \mu(x_*; \theta^{(i)})$ for $\theta^{(i)} = \text{MLE}(\mathcal{D}, \theta_0^{(i)})$
  this can disregard any spread the MLE solutions might have captured
- others will ensemble likelihood assessments:
  $\frac{1}{K}\sum_{i=1}^{K} p(y_*|x_*, \theta^{(i)})$ for $\theta^{(i)} = \text{MLE}(\mathcal{D}, \theta_0^{(i)})$
  this requires a prediction a specific query y_*

I want to say *yes*, because if we are going to talk about ‘ensembles’, then I feel like I owe a word such as ‘fancy’ to Bayes.

But honestly, I don’t feel like using the word ‘fancy’, because marginal and conditional probabilities are pretty basic.

Finally, I’m not sure it’s fair to explain something self-consistent (i.e., probability calculus) in terms of such a vaguely specified notion (i.e., ensembling). Perhaps we should explain ensembling as an attempt at probability calculus?

But let’s build intuition, and expose differences!

- Contrast this with stochasticity due to assessments of $\mu(x_*; \theta)$ or $p(y_*|x_*, \theta)$ for $\theta \sim \Theta|\mathcal{D}, \alpha$

Bayesian Reasoning

Reasoning with *parametric BNNs* involves averaging over parameters

- in a Bayesian sense a *model* is a set of assumptions
  e.g. conditional independences, choice of prior, choice of likelihood, architecture blocks, hyperparameters
- formally we should write $p(\mathcal{D}, \theta|\alpha, \mathcal{M})$ where $\mathcal{M}$ is the model assumptions — we omit $\mathcal{M}$ for brevity
- θ is only one *hypothesis* under this model
- the "worth" of each θ is quantified by $p(\theta|\mathcal{D}, \alpha)$
- uncertainty estimates are based on this posterior probability

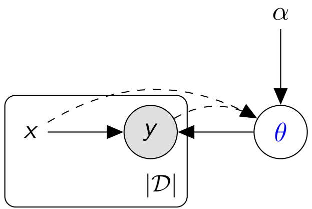
Probabilistic inference

- marginalisation
- conditioning on observations

is the key to Bayes.

Bayesian Inference

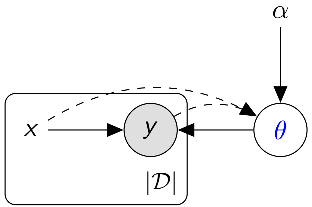
Posterior inference



- The graphical model tells us that observations are independent of one another given θ. The dashed arrows illustrate posterior inference, they show that if we were to condition on observations, the posterior distribution would depend on **all** data points.

Bayesian Inference

Posterior inference

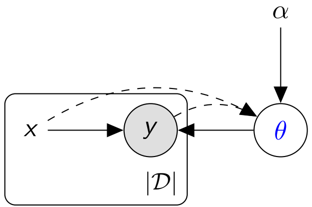


$$p(\theta|\mathcal{D}) = \frac{\overbrace{p(\theta)}^{\text{prior}}\,\overbrace{p(\mathcal{D}|\theta)}^{\text{likelihood}}}{\underbrace{p(\mathcal{D})}_{\text{evidence}}}$$

- The graphical model tells us that observations are independent of one another given θ. The dashed arrows illustrate posterior inference, they show that if we were to condition on observations, the posterior distribution would depend on **all** data points.
- This is also clear from the fact that the evidence enters the computation of a posterior probability.

Bayesian Inference

Posterior inference



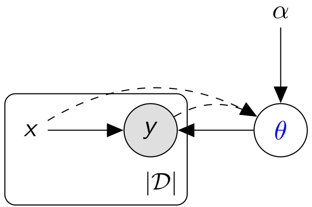
$$p(\theta|\mathcal{D}) = \frac{\overbrace{p(\theta)}^{\text{prior}}\,\overbrace{p(\mathcal{D}|\theta)}^{\text{likelihood}}}{\underbrace{p(\mathcal{D})}_{\text{evidence}}}$$

$$p(\mathcal{D}|\theta) = \prod_{\langle x,y\rangle \in \mathcal{D}} p(y|x,\theta)$$

- The graphical model tells us that observations are independent of one another given θ. The dashed arrows illustrate posterior inference, they show that if we were to condition on observations, the posterior distribution would depend on **all** data points.
- This is also clear from the fact that the evidence enters the computation of a posterior probability.
- Whereas likelihood is straightforward to assess.

Bayesian Inference

Posterior inference



$$p(\theta|\mathcal{D}) = \frac{\overbrace{p(\theta)}^{\text{prior}}\overbrace{p(\mathcal{D}|\theta)}^{\text{likelihood}}}{\underbrace{p(\mathcal{D})}_{\text{evidence}}}$$

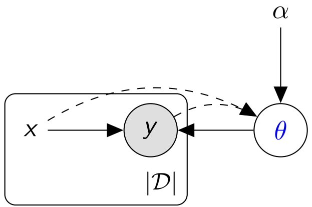
$$p(\mathcal{D}|\theta) = \prod_{\langle x,y\rangle \in \mathcal{D}} p(y|x,\theta)$$

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

- The graphical model tells us that observations are independent of one another given θ. The dashed arrows illustrate posterior inference, they show that if we were to condition on observations, the posterior distribution would depend on **all** data points.
- This is also clear from the fact that the evidence enters the computation of a posterior probability.
- Whereas likelihood is straightforward to assess.
- The evidence takes integrating over the entire sample space of Θ.

Bayesian Inference

Posterior inference

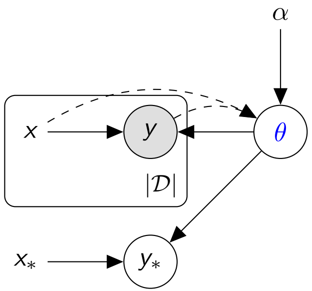


$$p(\theta|\mathcal{D}) = \frac{\overbrace{p(\theta)}^{\text{prior}}\ \overbrace{p(\mathcal{D}|\theta)}^{\text{likelihood}}}{\underbrace{p(\mathcal{D})}_{\text{evidence}}}$$

$$p(\mathcal{D}|\theta) = \prod_{\langle x,y\rangle\in\mathcal{D}} p(y|x,\theta)$$

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

Posterior predictive distribution

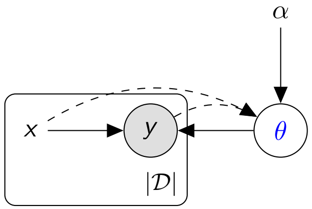


$$p(y_*|x_*,\mathcal{D}) =$$

- The graphical model tells us that observations are independent of one another given θ. The dashed arrows illustrate posterior inference, they show that if we were to condition on observations, the posterior distribution would depend on **all** data points.
- This is also clear from the fact that the evidence enters the computation of a posterior probability.
- Whereas likelihood is straightforward to assess.
- The evidence takes integrating over the entire sample space of Θ.
- Let's see what happens when we have a novel input x_*, whose response y_* we don't know.

Bayesian Inference

Posterior inference

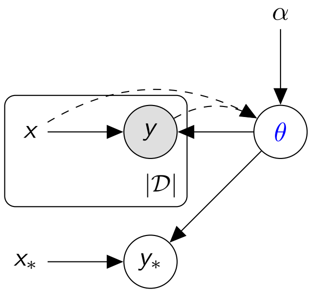


$$p(\theta|\mathcal{D}) = \frac{\overbrace{p(\theta)}^{\text{prior}}\,\overbrace{p(\mathcal{D}|\theta)}^{\text{likelihood}}}{\underbrace{p(\mathcal{D})}_{\text{evidence}}}$$

$$p(\mathcal{D}|\theta) = \prod_{\langle x,y\rangle\in\mathcal{D}} p(y|x,\theta)$$

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

Posterior predictive distribution

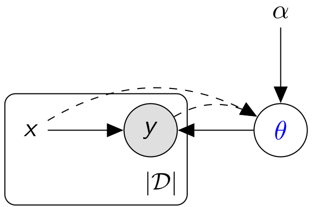


$$p(y_*|x_*,\mathcal{D}) = \int p(y_*,\theta|\mathcal{D},x_*)\mathrm{d}\theta$$

- The graphical model tells us that observations are independent of one another given θ. The dashed arrows illustrate posterior inference, they show that if we were to condition on observations, the posterior distribution would depend on **all** data points.
- This is also clear from the fact that the evidence enters the computation of a posterior probability.
- Whereas likelihood is straightforward to assess.
- The evidence takes integrating over the entire sample space of Θ.
- Let's see what happens when we have a novel input x_*, whose response y_* we don't know.
- The marginal probability for any value y_* in the response space $\mathcal{Y}$ takes marginalising θ out.

Bayesian Inference

Posterior inference

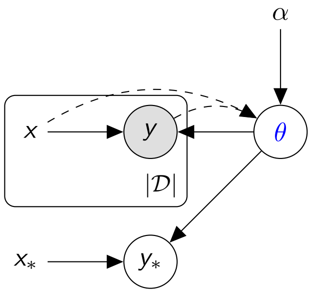


$$p(\theta|\mathcal{D}) = \frac{\overbrace{p(\theta)}^{\text{prior}}\,\overbrace{p(\mathcal{D}|\theta)}^{\text{likelihood}}}{\underbrace{p(\mathcal{D})}_{\text{evidence}}}$$

$$p(\mathcal{D}|\theta) = \prod_{\langle x,y\rangle\in\mathcal{D}} p(y|x,\theta)$$

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

Posterior predictive distribution



$$\begin{aligned} p(y_*|x_*,\mathcal{D}) &= \int p(y_*,\theta|\mathcal{D},x_*)\mathrm{d}\theta \\ &= \int p(\theta|\mathcal{D})p(y_*|x_*,\theta)\mathrm{d}\theta \end{aligned}$$

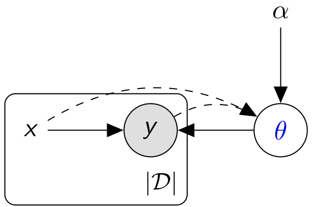
Conditional independence

$Y_* \perp \mathcal{D} \mid \theta$

- The graphical model tells us that observations are independent of one another given θ. The dashed arrows illustrate posterior inference, they show that if we were to condition on observations, the posterior distribution would depend on **all** data points.
- This is also clear from the fact that the evidence enters the computation of a posterior probability.
- Whereas likelihood is straightforward to assess.
- The evidence takes integrating over the entire sample space of Θ.
- Let's see what happens when we have a novel input x_*, whose response y_* we don't know.
- The marginal probability for any value y_* in the response space $\mathcal{Y}$ takes marginalising θ out.
- Our graphical model shows clearly that given θ the likelihood $p(y_*|x_*,\theta)$ is independent of the data $\mathcal{D}$. This reveals the posterior predictive probability: the expected value of the likelihood under the posterior, i.e., $\mathbb{E}_{\Theta|\mathcal{D}}[p(y_*|x_*,\Theta)]$. This is also known as Bayesian averaging.

Bayesian Inference

Posterior inference

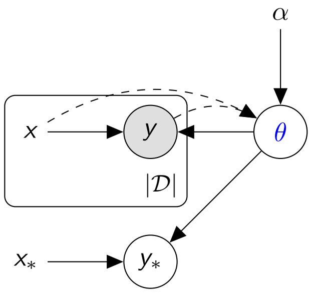


$$p(\theta|\mathcal{D}) = \frac{\overbrace{p(\theta)}^{\text{prior}}\,\overbrace{p(\mathcal{D}|\theta)}^{\text{likelihood}}}{\underbrace{p(\mathcal{D})}_{\text{evidence}}}$$

$$p(\mathcal{D}|\theta) = \prod_{\langle x,y\rangle \in \mathcal{D}} p(y|x,\theta)$$

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

Posterior predictive distribution



$$\begin{aligned} p(y_*|x_*,\mathcal{D}) &= \int p(y_*,\theta|\mathcal{D},x_*)\mathrm{d}\theta \\ &= \int p(\theta|\mathcal{D})p(y_*|x_*,\theta)\mathrm{d}\theta \end{aligned}$$

Conditional independence

$Y_* \perp \mathcal{D} \mid \theta$

- The graphical model tells us that observations are independent of one another given θ. The dashed arrows illustrate posterior inference, they show that if we were to condition on observations, the posterior distribution would depend on **all** data points.
- This is also clear from the fact that the evidence enters the computation of a posterior probability.
- Whereas likelihood is straightforward to assess.
- The evidence takes integrating over the entire sample space of Θ.
- Let's see what happens when we have a novel input x_*, whose response y_* we don't know.
- The marginal probability for any value y_* in the response space $\mathcal{Y}$ takes marginalising θ out.
- Our graphical model shows clearly that given θ the likelihood $p(y_*|x_*,\theta)$ is independent of the data $\mathcal{D}$. This reveals the posterior predictive probability: the expected value of the likelihood under the posterior, i.e., $\mathbb{E}_{\Theta|\mathcal{D}}[p(y_*|x_*,\Theta)]$. This is also known as Bayesian averaging.

Terminology

Prior	$p(\theta)$
Likelihood	$p(\mathcal{D}\|\theta)$
Posterior	$p(\theta\|\mathcal{D})$
Evidence (aka Marginal Likelihood)	$p(\mathcal{D})$
Posterior predictive distribution	$p(y_*\|x_*, \mathcal{D})$
Probabilistic inference	marginalisation/expectation/ conditioning on observations

Remark: in DL the word *inference* is used differently, it usually has to do with assessing a decision rule.

Summary

BNNs are NNs with priors over parameters

The goal is to take uncertainty seriously

Uncertainty estimates help make decisions, e.g.

- model comparison and selection
- when a human should intervene

Other uses include

- reinforcement learning
- active learning
- meta-learning
- learn from streaming data

Bayesian reasoning requires probabilistic inference

Literature

BDA3 Gelman et al. (2013)

- Chapter 1 for an overview of Bayesian theory and useful terminology
- All of Part I for an introductory course
- Chapter 4 for connections to non-Bayesian approaches

BDA3 is now available for free: http://www.stat.columbia.edu/~gelman/book/

Outline

1. Bayes: what and why?
2. Choosing a prior
3. Posterior Inference for BNNs
4. Bayesian Dropout
5. Example

How does one choose a prior?

A prior is meant to capture our beliefs about the phenomenon we are modelling – in this case the relationship between x and y

Let's first consider a simple example: mixture model

$$Z|\pi \sim \mathrm{Cat}(\pi)$$
$$X|\theta, z \sim \mathrm{Cat}(\theta^{(z)})$$

We first select a discrete mixture component z, this component then selects a Categorical distribution from which we generate a data point x

MLE

$$\pi = {}^{1}/_{K}\mathbf{1}_K$$
$$\theta = \langle \theta^{(1)}, \ldots, \theta^{(K)} \rangle$$

Bayes

$$\pi|\alpha \sim \mathrm{Dir}(\alpha\mathbf{1}_K)$$
$$\theta^{(k)}|\beta \sim \mathrm{Dir}(\beta\mathbf{1}_V)$$

$\mathbf{1}_K$ is a K-dimensional vector where every element is 1.

In **MLE** parameters are **given**. Where do they come from? Usually something like $\arg\max_\theta \log p(\mathcal{D}|\theta)$.

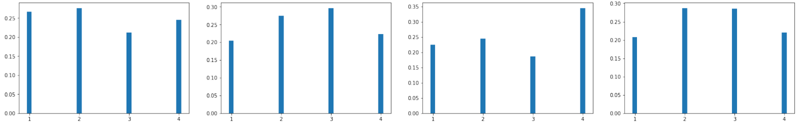
For **Bayes** parameters are rvs (there's no search for parameters, they come from the prior, stochastically). Let me emphasise this: **there is no search**. Learning is not the task of finding the model parameters that suits a criterion. That is what learning comes down to in terms of MLE, but that's not what learning needs to be generally. In Bayesian inference, we start from some general ideas about θ, coded in $\Theta|\alpha$, and updated our beliefs by inferring $\Theta|\mathcal{D}, \alpha$. Using the Bayesian model to make decisions **dispenses with** ever singling out any 'optimum' θ.

It is important to ask: what does it mean

- to impose a Dirichlet prior on mixing coefficients π?
- to impose a Dirichlet prior on the parameters of each likelihood component $\theta^{(z)}$?

What makes good mixing coefficients?

Say we have $K = 4$ components, I show a few samples for $\pi \sim \text{Dir}(10 \times \mathbf{1}_K)$



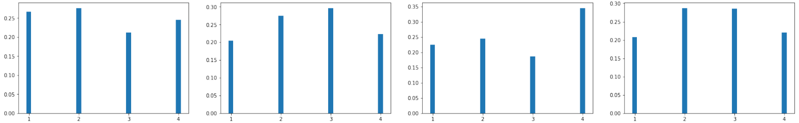
The question you have to think about is: Can we make any assumptions before observing data?

For example, let's see what happens as we vary our choice of prior for mixing coefficients

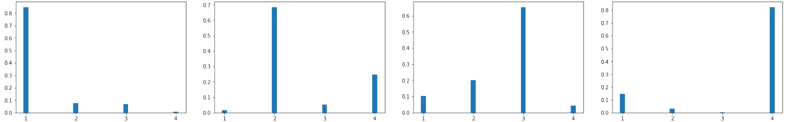
- Here you see 4 models that are likely under this prior. Each of these models gives every component of the mixture roughly the same opportunity to generate data points.

What makes good mixing coefficients?

Say we have $K = 4$ components, I show a few samples for $\pi \sim \text{Dir}(10 \times \mathbf{1}_K)$



$\pi \sim \text{Dir}(1 \times \mathbf{1}_K)$



The question you have to think about is: Can we make any assumptions before observing data?

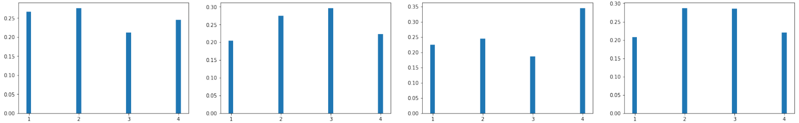
For example, let's see what happens as we vary our choice of prior for mixing coefficients

- Here you see 4 models that are likely under this prior. Each of these models gives every component of the mixture roughly the same opportunity to generate data points.
- This prior is different, each model prefers few components. Clearly, there's no reason to prefer component 1 systematically over the others, so this prior supports coefficients that reveal different winners each time.

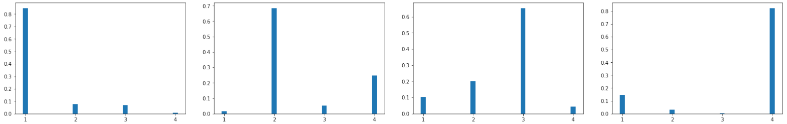
What makes good mixing coefficients?

Say we have $K = 4$ components, I show a few samples for

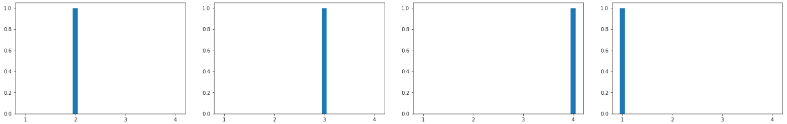
$\pi \sim \mathrm{Dir}(10 \times \mathbf{1}_K)$



$\pi \sim \mathrm{Dir}(1 \times \mathbf{1}_K)$



$\pi \sim \mathrm{Dir}(0.1 \times \mathbf{1}_K)$



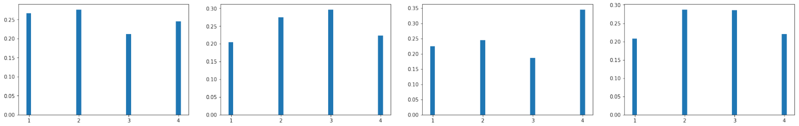
The question you have to think about is: Can we make any assumptions before observing data?

For example, let's see what happens as we vary our choice of prior for mixing coefficients

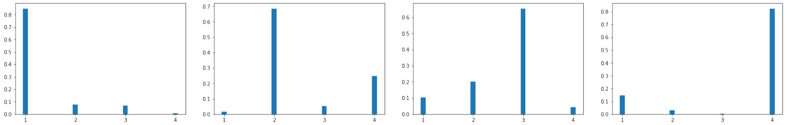
- Here you see 4 models that are likely under this prior. Each of these models gives every component of the mixture roughly the same opportunity to generate data points.
- This prior is different, each model prefers few components. Clearly, there's no reason to prefer component 1 systematically over the others, so this prior supports coefficients that reveal different winners each time.
- This prior likes very sparse mixing coefficients. Again, there's no reason to prefer any one component over any other. But it seems like a good idea to prefer sparse mixtures. The Dirichlet prior allows us to express a preference: every model is possible (literally, every way to mix for components), but some models are preferred (for example, the sparse ones).

What makes good mixing coefficients?

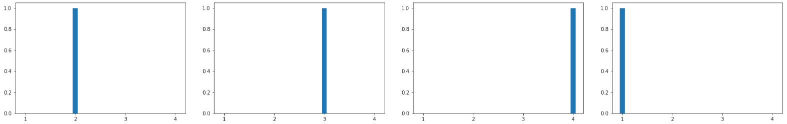
Say we have $K = 4$ components, I show a few samples for $\pi \sim \mathrm{Dir}(10 \times \mathbf{1}_K)$



$\pi \sim \mathrm{Dir}(1 \times \mathbf{1}_K)$



$\pi \sim \mathrm{Dir}(0.1 \times \mathbf{1}_K)$



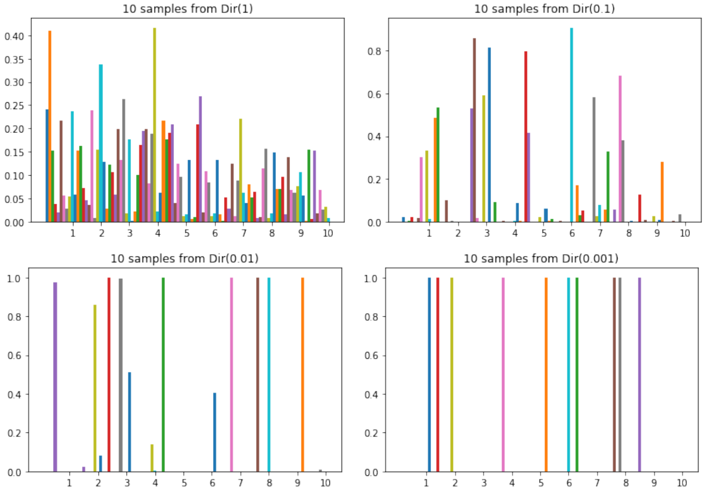
The question you have to think about is: Can we make any assumptions before observing data?

For example, let's see what happens as we vary our choice of prior for mixing coefficients

- Here you see 4 models that are likely under this prior. Each of these models gives every component of the mixture roughly the same opportunity to generate data points.
- This prior is different, each model prefers few components. Clearly, there's no reason to prefer component 1 systematically over the others, so this prior supports coefficients that reveal different winners each time.
- This prior likes very sparse mixing coefficients. Again, there's no reason to prefer any one component over any other. But it seems like a good idea to prefer sparse mixtures. The Dirichlet prior allows us to express a preference: every model is possible (literally, every way to mix for components), but some models are preferred (for example, the sparse ones).

What makes a good conditional?

Say we have $V = 10$ types of data points, I show samples $\theta^{(k)}|\beta \sim \text{Dir}(\beta)$



10 samples from Dir(1)

10 samples from Dir(0.1)

10 samples from Dir(0.01)

10 samples from Dir(0.001)

How about the components themselves? Similar story.

Do we want components that each can generate everything? That is, no specialisation whatsoever.

Or do we prefer components that generate only a selected subset of the support?

Clearly, why should component 1 prefer any particular subset of the support? It should not, thus it's prior does not express a preference for one particular subset, it express a preference for any distribution that focuses on only a small subset. Distributions that meet this 'requirement' are not preferred over one another.

The idea of allowing components to change like that may be scary. But think of it this way, we are not committing to any one such distribution. We are averaging **all** of them out to get to quantities such as the evidence and the posterior predictive distribution. There's no risk in that. The risk is precisely in arbitrarily picking any one configuration when so many alternatives exist.

Mixture Models are Simple to Understand

The unobservable random variables π and $\theta^{(k)}$ are rather interpretable

- it's clear that we want assignments to be unambiguous
  sparse mixing weights
- it's clear that we want components to be rather selective
  sparse conditionals
- it's clear that we don't know the identity of clusters
  uniform marginals

All of that is essentially very clear a priori

- that is, before we collect observations
- by simply considering the nature of problem

Yes, there's an elephant in the room.

Mixture Models are Simple to Understand

The unobservable random variables π and $\theta^{(k)}$ are rather interpretable

- it's clear that we want assignments to be unambiguous
  sparse mixing weights
- it's clear that we want components to be rather selective
  sparse conditionals
- it's clear that we don't know the identity of clusters
  uniform marginals

All of that is essentially very clear a priori

- that is, before we collect observations
- by simply considering the nature of problem

Meaning of weights in NNs are quite obscure! Who can tell what aspect of a classifier any of the LSTM parameters controls?

Yes, there's an elephant in the room.

Learning Functions

We are essentially using NNs to learn some unknown function that maps from data to probabilities — which then support decisions

Learning Functions

We are essentially using NNs to learn some unknown function that maps from data to probabilities — which then support decisions

It's only natural to ask, *What functions can NNs learn*?

- usually one says "any continuous and deterministic function" (Funahashi, 1989), given a wide enough hidden layer

Learning Functions

We are essentially using NNs to learn some unknown function that maps from data to probabilities — which then support decisions

It's only natural to ask, *What functions can NNs learn*?

- usually one says "any continuous and deterministic function" (Funahashi, 1989), given a wide enough hidden layer

Fair, but how about this,

- *What functions can we actually recover given finite observations?*
- *How about the fact that we employ convex optimisers?*

It's hard to talk about what functions we can learn when the most important factors are *amount of data* and the *success of a local optimiser*

Random functions

Let's consider what happens when our parameters are random following a given prior.

Sampling from these priors and performing forward passes with the network will expose a range of functions

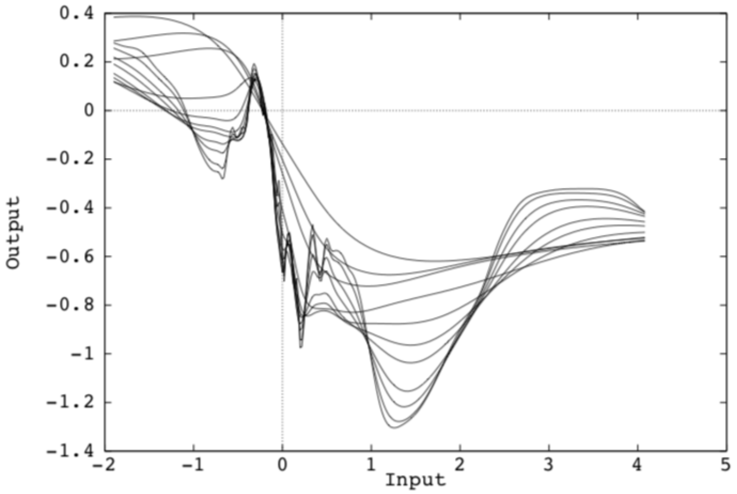
- some might have specific properties
  e.g. smoothness, periodicity
- some will be preferred over others
- some may be impossible
  e.g. Brownian functions vs infinitely smooth functions

NN architectures carefully incorporate a number of inductive biases (think about it: how does the LSTM overcome limitations of vanilla RNNs? how does the Transformer get rid of the bottleneck of recurrent architectures? how does a CNN achieve spatial invariance?).

A prior over any NN parameters, even a vague prior, induces a highly structured prior over functions.

Try this yourself: pick an NN block you like and plot $f(x;\theta)$ for $x \in \mathbb{R}$ for different samples θ from a prior. How about this, get a simple prior that essentially only expresses a preference for numbers of small magnitude (e.g., $\mathcal{U}(-\sigma, +\sigma)$ or $\mathcal{N}(0, \sigma^2)$).

Draws



Example from MacKay (1998)

MacKay (1998) tries the experiment we just sketched, I highly recommend you check his examples.

In this example, he shows that he can express preferences over how smooth a function is.

Role of Priors

Priors encode assumptions we can make prior to observing data.

But more than that, priors are a pre-requisite for probabilistic inference: we need a joint distribution to be able to marginalise parameters and/or condition on observations.

As BNNs are distributions over functions, it's often difficult to make assumptions about their parameters, we should instead focus on the distribution over functions they induce.

To gain some intuition, let us look into a known prior over functions.

Where we don't know much a priori, non-informative and weakly informative priors can be designed (Gelman et al., 2013, Section 2.8).

Though see the difference: a vague prior over parameters need not specify a vague prior over functions, because the mapping from θ to $f(x;\theta)$ is hand-crafted and carefully designed (a worthy NN architecture is not an arbitrary stack of differentiable transformations).

Gaussian Processes Prior

Consider the case of regression, where $y = f(x) + \epsilon$ for some $\epsilon \sim \mathcal{N}(0, \tau^{-1})$

- this implies $Y|f(x) \sim \mathcal{N}(f(x), \tau^{-1})$
- let's design a prior for $f(x)$
  Note that a parametric way to do so is to say $f(x) = w^\top \phi(x)$ for some fixed feature function $\phi(x)$ and impose a prior on w, but then again, what are the properties of such a prior?

The probability distribution of a function $f(x)$ is a Gaussian process (GP) if for any finite selection of points $x^{(1)}, \ldots, x^{(N)}$ the density $p(f(x^{(1)}), \ldots, f(x^{(N)}))$ is a Gaussian.

A function represents an infinite object, but in ML we typically only reason over finite datasets!

It's not crucial, for this course, that you understand a Gaussian processes, but it does help motivate BNNs.

The GP is a prior over functions (these functions can be real-valued or vector-valued).

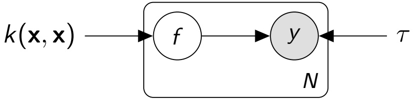
We have a GP if for a finite number of evaluations of the function, that is, $\mathbf{f} = \{f(x^{(1)}), \ldots, f(x^{(N)})\}$ for inputs $\mathbf{x} = \{x^{(1)}, \ldots, x^{(N)}\}$, the joint distribution $\mathbf{F}|\mathbf{x}$ is a multivariate Gaussian. See that the function evaluations are themselves rvs (and I've gathered a collection of inputs and a collection of function evaluations in boldfaced variables, for brevity).

Recall that to specify such a multivariate Gaussian we need to specify an N-dimensional mean vector, and a $N \times N$ covariance matrix (not an arbitrary matrix though, do you remember the constraints that apply?).

GP Regression

I'll employ boldfacing to denote a collection of N datapoints, e.g. $\mathbf{x} = \{x^{(1)}, \ldots, x^{(N)}\}$ and $\mathbf{y} = \{y^{(1)}, \ldots, y^{(N)}\}$, indexing returns an element, e.g. $x_i \overset{\text{def}}{=} x^{(i)}$. Similarly, $\mathbf{f} = \{f(x^{(1)}), \ldots, f(x^{(N)})\}$ is a collection of latent function assessments.

$$\mathbf{F}|\mathbf{x} \sim \mathcal{N}(\mathbf{0}, k(\mathbf{x}, \mathbf{x}))$$
$$\mathbf{Y}|\mathbf{f} \sim \mathcal{N}(\mathbf{f}, \tau^{-1}\mathbf{I}_N)$$



The covariance matrix is defined by a kernel function $k(x, x')$

- I abuse notation and use $k(\mathbf{x}, \mathbf{x})$ to denote the $N \times N$ matrix $\mathbf{K}$ of kernel assessments, i.e. $K_{i,j} = k(x_i, x_j)$
- $k(x', \mathbf{x})$ denotes a row-vector of kernel assessments

Check the *excellent* Kernel Cookbook by David Duvenaud

We assume the GP prior has 0 mean and we specify an $N \times N$ covariance matrix by comparing x_i to x_j using a *kernel function*, which I denote by $k(x, x')$.

The graphical model shows clearly that we assume the function to be a latent variable.

The GP is our choice of prior over functions, as usual, specifying a probabilistic model still requires a choice of likelihood. Let's concentrate on a regression problem and pick a Gaussian likelihood.

Our choices are: the kernel function, and the likelihood (which will depend on the type of data we model).

Avoid confusions: a GP has nothing to do with your function looking Gaussian, we are talking about N evaluations of your function (which is allowed to take many many many forms) being random outcomes that are jointly distributed by a multivariate Gaussian.

Conjugate Inference for GP regression

What's the family of the marginal of a GP model for some given $\mathbf{x}, \mathbf{y}$?

$$p(\mathbf{y}|\mathbf{x}) = \int p(\mathbf{y}, \mathbf{f}|\mathbf{x})\mathrm{d}\mathbf{f} =$$

Check Bishop (2006, Chapter 2) for operations with Multivariate Gaussians.

Conjugate Inference for GP regression

What's the family of the marginal of a GP model for some given $\mathbf{x}, \mathbf{y}$?

$$p(\mathbf{y}|\mathbf{x}) = \int p(\mathbf{y}, \mathbf{f}|\mathbf{x})\mathrm{d}\mathbf{f} = \int \underbrace{\mathcal{N}(\mathbf{f}|\mathbf{0}, k(\mathbf{x}, \mathbf{x}))\mathcal{N}(\mathbf{y}|\mathbf{f}, \tau^{-1}\mathbf{I}_N)}_{\text{jointly Gaussian}}\,\mathrm{d}\mathbf{f}$$

Recall: marginals of a multivariate Gaussian are Gaussians!

Check Bishop (2006, Chapter 2) for operations with Multivariate Gaussians.

Conjugate Inference for GP regression

What's the family of the marginal of a GP model for some given $\mathbf{x}, \mathbf{y}$?

$$p(\mathbf{y}|\mathbf{x}) = \int p(\mathbf{y}, \mathbf{f}|\mathbf{x}) \mathrm{d}\mathbf{f} = \int \underbrace{\mathcal{N}(\mathbf{f}|\mathbf{0}, k(\mathbf{x}, \mathbf{x}))\mathcal{N}(\mathbf{y}|\mathbf{f}, \tau^{-1}\mathbf{I}_N)}_{\text{jointly Gaussian}} \mathrm{d}\mathbf{f}$$

Recall: marginals of a multivariate Gaussian are Gaussians!

Thus what's the family of the posterior?

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

Check Bishop (2006, Chapter 2) for operations with Multivariate Gaussians.

Conjugate Inference for GP regression

What's the family of the marginal of a GP model for some given $\mathbf{x}, \mathbf{y}$?

$$p(\mathbf{y}|\mathbf{x}) = \int p(\mathbf{y}, \mathbf{f}|\mathbf{x}) \mathrm{d}\mathbf{f} = \int \underbrace{\mathcal{N}(\mathbf{f}|\mathbf{0}, k(\mathbf{x}, \mathbf{x}))\mathcal{N}(\mathbf{y}|\mathbf{f}, \tau^{-1}\mathbf{I}_N)}_{\text{jointly Gaussian}} \mathrm{d}\mathbf{f}$$

Recall: marginals of a multivariate Gaussian are Gaussians!

Thus what's the family of the posterior?

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

Recall: conditioning on a subset of jointly Gaussian variables yields a multivariate Gaussian

Check Bishop (2006, Chapter 2) for operations with Multivariate Gaussians.

Exact Inference with GPs

Posterior

$$
\begin{aligned}
\mathbf{F}|\mathbf{x}, \mathbf{y} &\sim \mathcal{N}(\mathbf{m}_{\text{post}}, \mathbf{K}_{\text{post}}) \\
\mathbf{m}_{\text{post}} &= \mathbf{K}(\mathbf{K} + \tau^{-1}\mathbf{I}_N)^{-1}\mathbf{y} \\
\mathbf{K}_{\text{post}} &= \mathbf{K} - \mathbf{K}(\mathbf{K} + \tau^{-1}\mathbf{I}_N)^{-1}\mathbf{K}^\top
\end{aligned}
$$

Posterior predictive distribution:

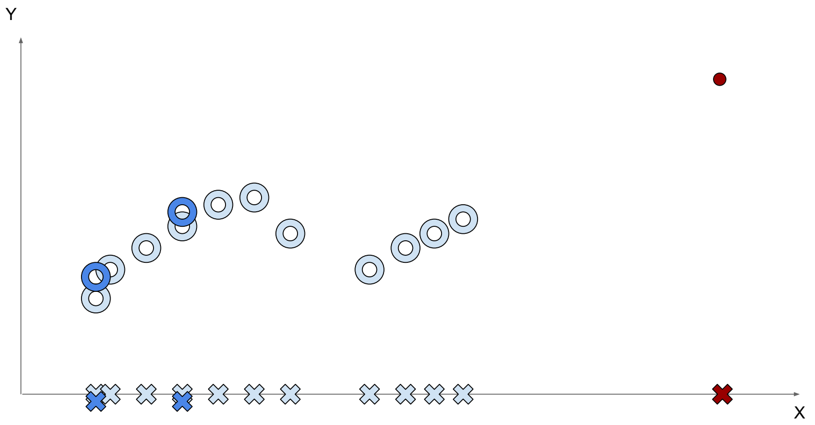
$$
\begin{aligned}
Y_*|x_*, \mathbf{x}, \mathbf{y} \sim \mathcal{N}(&k(x_*, \mathbf{x})(\mathbf{K} + \tau^{-1}\mathbf{I}_N)^{-1}\mathbf{y}, \\
&k(x_*, x_*) + \tau^{-1} - k(x_*, \mathbf{x})(\mathbf{K} + \tau^{-1}\mathbf{I}_N)^{-1}k(x_*, \mathbf{x})^\top)
\end{aligned}
$$

The key here is not that you memorise these expressions, the point is that this involves no more than kernel assessments (to evaluate **K** and $k(x_*, \mathbf{x})$) and a bit of linear algebra.

This result is truly remarkable. We can reason about a novel input x_* using **all** latent functions that are likely given our observations, and these functions can be very flexible (with general properties controlled by our choice of kernel function), and all we need to do is a bit of linear algebra. Not particularly scalable (in N) linear algebra, but still, we are talking about **all** infinitely many functions in the support of our GP prior.

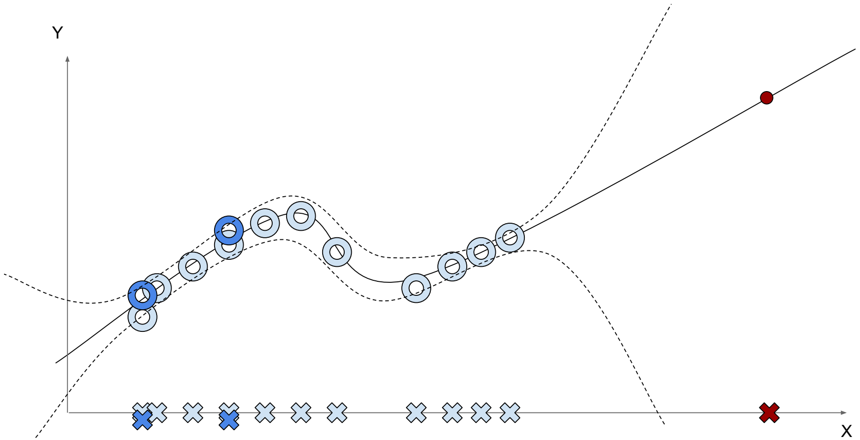
Do you see that there is no search (optimisation)?

Uncertainty illustrated (revisited)



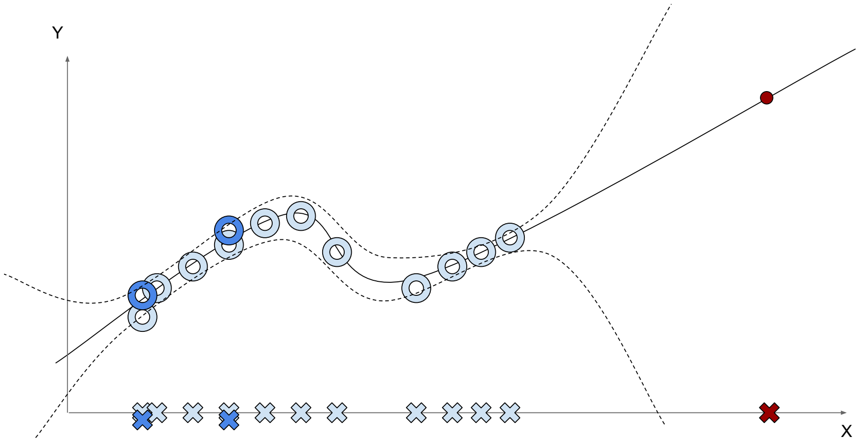
Let's get back to this

Uncertainty illustrated (revisited)



What if uncertainty depended on the distance to observations?

Uncertainty illustrated (revisited)



Kernels in GPs operationalise this notion

Terminology

Random functions: latent treatment to $f(x)$

Kernel: $k : \mathcal{X} \times X \to \mathbb{R}$ such that $k(x, x')$ is the covariance between $f(x)$ and $f(x')$

Gaussian process prior: $\mathbf{F}|\mathbf{x} \sim \mathcal{G}P(0, k(\mathbf{x}, \mathbf{x}))$

Conjugate GP inference: with Gaussian likelihood, marginals and conditionals are Gaussians

Summary

NNs specify functions.

BNNs learn a distribution over such functions by treating parameters as random variables.

The effect of a parameter over the learned function is not obvious, and NN functions encode rich inductive biases.

A vague prior over NN parameters still induces a structured distribution over functions.

A prior over functions can be specified in a non-parametric way via specification of a covariance (kernel) function.

A GP prior is a well-studied structured prior over functions

Literature

David MacKay's pioneering work

Bayesian interpolation	MacKay (1992a)
or go all the way through his PhD thesis	MacKay (1992b)
Priors for Infinite Networks	Neal (1994, 1996)
Multivariate Gaussians	Bishop (2006, Chapter 2)
Introduction to GPs	MacKay (1998)

GP summer school classes by Neil Laurence

Kernel Cookbook by David Duvenaud

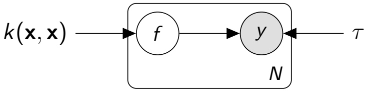
Neal (1994): `https://www.cs.toronto.edu/~radford/ftp/pin.pdf`

Outline

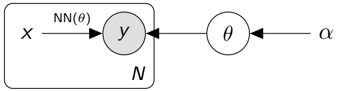
1. Bayes: what and why?
2. Choosing a prior
3. **Posterior Inference for BNNs**
4. Bayesian Dropout
5. Example

GPs vs BNNs

GP



BNN



GP's a non-parametric models

- the complexity (or capacity) of the model grows with the data
- posterior predictive is known and tractable
- we know a lot about the random functions we get

BNNs are parametric models

- the complexity (or capacity) is pre-specified
- posterior predictive is unknown and intractable
- we know little about the random functions we get

See that the GP prior depends on **all** covariates.

The BNN prior is specified directly over NN parameters.

A BNN can be thought of as a parametric approximation to a GP, where instead of a fixed kernel we have a parametrised kernel and a distribution over the parameters of the kernel. More on this in the next few slides.

Why don't we always use GPs then?

Flexibility

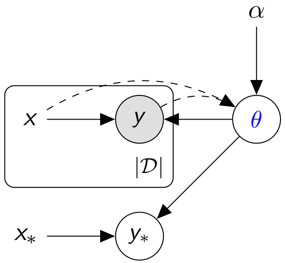
- kernels for text are fewer, less convenient, and less well-understood
- x can be very high-dimensional (and perhaps we have less intuitions to choose a kernel)
- conjugate inference is only possible with Gaussian likelihood

Computational complexity

- exact GP inference takes $O(N^3)$
- it's possible to scale them up, but that's an active research topic
- many solutions are specific to continuous inputs

Let’s then consider Bayesian inference for a BNN

This is essentially what we have to address



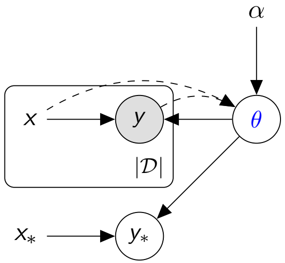
Forget GPs for a moment, let’s talk about BNNs. Bayesian inference is not about a particular model, after all.

So, let’s just look for the posterior predictive distribution of a BNN!

Clearly, cannot be tractable! But wait, do we know a blackbox algorithm to address intractable inferences?

Let’s then consider Bayesian inference for a BNN

This is essentially what we have to address



That is,

$$p(y_*|\mathcal{D},\alpha,x_*) = \int p(y_*,\theta|\mathcal{D},\alpha)\mathrm{d}\theta = \int p(\theta|\mathcal{D},\alpha)p(y_*|\theta,x_*)\mathrm{d}\theta$$

But $p(\theta|\mathcal{D},\alpha) = \int p(\theta|\alpha)p(\mathcal{D}|\theta)\mathrm{d}\theta$ is intractable

Forget GPs for a moment, let’s talk about BNNs. Bayesian inference is not about a particular model, after all.

So, let’s just look for the posterior predictive distribution of a BNN!

Clearly, cannot be tractable! But wait, do we know a blackbox algorithm to address intractable inferences?

Variational Bayes

Let's learn a proxy $q(\theta|\lambda)$ to $p(\theta|\mathcal{D})$ and solve

$$p(y_*, \theta|\mathcal{D}, x_*) = \int p(\theta|\mathcal{D}) p(y_*|\theta, x_*) \mathrm{d}\theta \approx \int q(\theta|\lambda) p(y_*|\theta, x_*) \mathrm{d}\theta$$

An alternative with guarantees is MCMC – as discussed in ML2. Example: MCMC for a mixture of Gaussians by David Blei.

Variational Bayes is the variational inference you know and love, where as good Bayesians we only optimise our choice of $q(\theta)$, not our choice of $p(\mathcal{D}, \theta)$.

Wait a second! Are we going to search/optimise? LoL, yes! But, we are going to optimise our approximations to intractable inferences, not the model itself.

Variational Bayes

Let's learn a proxy $q(\theta|\lambda)$ to $p(\theta|\mathcal{D})$ and solve

$$p(y_*, \theta|\mathcal{D}, x_*) = \int p(\theta|\mathcal{D})p(y_*|\theta, x_*)\mathrm{d}\theta \approx \int q(\theta|\lambda)p(y_*|\theta, x_*)\mathrm{d}\theta$$

Principle: choose an approximation that minimises KL-divergence

$$\begin{aligned}&\arg\min_{q(\theta)} \ \mathrm{KL}(q(\theta)||p(\theta|\mathcal{D}))\\ &= \arg\min_{q(\theta)} \ \mathbb{E}_{q(\theta)}\left[\log \frac{q(\theta|\lambda)}{p(\theta|\mathcal{D})}\right] \quad \text{definition of KL}\end{aligned}$$

An alternative with guarantees is MCMC – as discussed in ML2. Example: MCMC for a mixture of Gaussians by David Blei.

Variational Bayes is the variational inference you know and love, where as good Bayesians we only optimise our choice of $q(\theta)$, not our choice of $p(\mathcal{D}, \theta)$.

Wait a second! Are we going to search/optimise? LoL, yes! But, we are going to optimise our approximations to intractable inferences, not the model itself.

Evidence Lowerbound (ELBO)

Principle: choose an approximation that minimises KL-divergence

$$\begin{aligned} &\arg\min_{q(\theta)} \ \mathrm{KL}(q(\theta)||p(\theta|\mathcal{D})) \\ &= \arg\min_{q(\theta)} \ \mathbb{E}_{q(\theta)}\left[\log \frac{q(\theta|\lambda)}{p(\theta|\mathcal{D})}\right] && \text{definition of KL} \end{aligned}$$

You know this from our first class on latent variable models. I'm repeating it here for convenience and with the variables that are relevant for this class.

Note I omit α here and there (due to lack of space), I hope it's not confusing.

The important message: picking the best posterior approximation is exactly equivalent to optimising the ELBO, no approximations there. Note the difference between this and VAEs. In a VAE, we optimise our choice of model as well (we search for θ), and even though we would like (in a VAE) to pick θ that leads to maximum log-likelihood, we can only pick θ that maximises a lowerbound. In VB, we are not searching for θ, as it is given random treatment, rather we search for the optimum of the lowerbound w.r.t. $q(\theta)$. That optimum is exactly the optimum of the VI objective $\arg\min_{q(\theta)} \ \mathrm{KL}(q(\theta)||p(\theta|\mathcal{D}))$.

Evidence Lowerbound (ELBO)

Principle: choose an approximation that minimises KL-divergence

$$
\begin{aligned}
&\arg\min_{q(\theta)} \ \mathrm{KL}(q(\theta)||p(\theta|\mathcal{D})) \\
&= \arg\min_{q(\theta)} \ \mathbb{E}_{q(\theta)}\left[\log \frac{q(\theta|\lambda)}{p(\theta|\mathcal{D})}\right] && \text{definition of KL} \\
&= \arg\min_{q(\theta)} \ \mathbb{E}_{q(\theta)}\left[\log \frac{q(\theta|\lambda)p(\mathcal{D})}{p(\theta,\mathcal{D})}\right] && \text{definition of posterior}
\end{aligned}
$$

You know this from our first class on latent variable models. I'm repeating it here for convenience and with the variables that are relevant for this class.

Note I omit α here and there (due to lack of space), I hope it's not confusing.

The important message: picking the best posterior approximation is exactly equivalent to optimising the ELBO, no approximations there. Note the difference between this and VAEs. In a VAE, we optimise our choice of model as well (we search for θ), and even though we would like (in a VAE) to pick θ that leads to maximum log-likelihood, we can only pick θ that maximises a lowerbound. In VB, we are not searching for θ, as it is given random treatment, rather we search for the optimum of the lowerbound w.r.t. $q(\theta)$. That optimum is exactly the optimum of the VI objective $\arg\min_{q(\theta)} \ \mathrm{KL}(q(\theta)||p(\theta|\mathcal{D}))$.

Evidence Lowerbound (ELBO)

Principle: choose an approximation that minimises KL-divergence

$$
\begin{aligned}
&\arg\min_{q(\theta)} \ \mathrm{KL}(q(\theta) || p(\theta|\mathcal{D})) \\
&= \arg\min_{q(\theta)} \ \mathbb{E}_{q(\theta)}\left[\log \frac{q(\theta|\lambda)}{p(\theta|\mathcal{D})}\right] && \text{definition of KL} \\
&= \arg\min_{q(\theta)} \ \mathbb{E}_{q(\theta)}\left[\log \frac{q(\theta|\lambda)p(\mathcal{D})}{p(\theta, \mathcal{D})}\right] && \text{definition of posterior} \\
&= \arg\min_{q(\theta)} \ \mathbb{E}_{q(\theta)}\left[\log \frac{q(\theta|\lambda)}{p(\theta, \mathcal{D})}\right] + \log p(\mathcal{D}) && \text{constant}
\end{aligned}
$$

You know this from our first class on latent variable models. I'm repeating it here for convenience and with the variables that are relevant for this class.

Note I omit α here and there (due to lack of space), I hope it's not confusing.

The important message: picking the best posterior approximation is exactly equivalent to optimising the ELBO, no approximations there. Note the difference between this and VAEs. In a VAE, we optimise our choice of model as well (we search for θ), and even though we would like (in a VAE) to pick θ that leads to maximum log-likelihood, we can only pick θ that maximises a lowerbound. In VB, we are not searching for θ, as it is given random treatment, rather we search for the optimum of the lowerbound w.r.t. $q(\theta)$. That optimum is exactly the optimum of the VI objective $\arg\min_{q(\theta)} \ \mathrm{KL}(q(\theta)||p(\theta|\mathcal{D}))$.

Evidence Lowerbound (ELBO)

Principle: choose an approximation that minimises KL-divergence

$$
\begin{aligned}
&\arg\min_{q(\theta)} \ \mathrm{KL}(q(\theta)||p(\theta|\mathcal{D})) \\
&= \arg\min_{q(\theta)} \ \mathbb{E}_{q(\theta)}\left[\log \frac{q(\theta|\lambda)}{p(\theta|\mathcal{D})}\right] && \text{definition of KL} \\
&= \arg\min_{q(\theta)} \ \mathbb{E}_{q(\theta)}\left[\log \frac{q(\theta|\lambda)p(\mathcal{D})}{p(\theta, \mathcal{D})}\right] && \text{definition of posterior} \\
&= \arg\min_{q(\theta)} \ \mathbb{E}_{q(\theta)}\left[\log \frac{q(\theta|\lambda)}{p(\theta, \mathcal{D})}\right] + \log p(\mathcal{D}) && \text{constant} \\
&= \arg\min_{q(\theta)} \ -\mathbb{E}_{q(\theta)}\left[\log \frac{p(\theta, \mathcal{D})}{q(\theta|\lambda)}\right] && \text{property of log}
\end{aligned}
$$

You know this from our first class on latent variable models. I'm repeating it here for convenience and with the variables that are relevant for this class.

Note I omit α here and there (due to lack of space), I hope it's not confusing.

The important message: picking the best posterior approximation is exactly equivalent to optimising the ELBO, no approximations there. Note the difference between this and VAEs. In a VAE, we optimise our choice of model as well (we search for θ), and even though we would like (in a VAE) to pick θ that leads to maximum log-likelihood, we can only pick θ that maximises a lowerbound. In VB, we are not searching for θ, as it is given random treatment, rather we search for the optimum of the lowerbound w.r.t. $q(\theta)$. That optimum is exactly the optimum of the VI objective $\arg\min_{q(\theta)} \ \mathrm{KL}(q(\theta)||p(\theta|\mathcal{D}))$.

Evidence Lowerbound (ELBO)

Principle: choose an approximation that minimises KL-divergence

$$
\begin{aligned}
&\arg\min_{q(\theta)} \; \mathrm{KL}(q(\theta)||{\color{red}p(\theta|\mathcal{D})}) \\
&= \arg\min_{q(\theta)} \; \mathbb{E}_{q(\theta)}\left[\log \frac{q(\theta|\lambda)}{{\color{red}p(\theta|\mathcal{D})}}\right] && \text{definition of KL} \\
&= \arg\min_{q(\theta)} \; \mathbb{E}_{q(\theta)}\left[\log \frac{q(\theta|\lambda){\color{red}p(\mathcal{D})}}{p(\theta,\mathcal{D})}\right] && \text{definition of posterior} \\
&= \arg\min_{q(\theta)} \; \mathbb{E}_{q(\theta)}\left[\log \frac{q(\theta|\lambda)}{p(\theta,\mathcal{D})}\right] + {\color{red}\log p(\mathcal{D})} && \text{constant} \\
&= \arg\min_{q(\theta)} \; -\mathbb{E}_{q(\theta)}\left[\log \frac{p(\theta,\mathcal{D})}{q(\theta|\lambda)}\right] && \text{property of log} \\
&= \arg\max_{q(\theta)} \; \mathbb{E}_{q(\theta)}\left[\log p(\theta,\mathcal{D})\right] + \mathbb{H}(q(\theta)) && \text{ELBO}
\end{aligned}
$$

You know this from our first class on latent variable models. I'm repeating it here for convenience and with the variables that are relevant for this class.

Note I omit α here and there (due to lack of space), I hope it's not confusing.

The important message: picking the best posterior approximation is exactly equivalent to optimising the ELBO, no approximations there. Note the difference between this and VAEs. In a VAE, we optimise our choice of model as well (we search for θ), and even though we would like (in a VAE) to pick θ that leads to maximum log-likelihood, we can only pick θ that maximises a lowerbound. In VB, we are not searching for θ, as it is given random treatment, rather we search for the optimum of the lowerbound w.r.t. $q(\theta)$. That optimum is exactly the optimum of the VI objective $\arg\min_{q(\theta)} \; \mathrm{KL}(q(\theta)||{\color{red}p(\theta|\mathcal{D})})$.

Evidence Lowerbound (ELBO)

Principle: choose an approximation that minimises KL-divergence

$$
\begin{aligned}
&\underset{q(\theta)}{\arg\min}\ \ \mathrm{KL}(q(\theta)||p(\theta|\mathcal{D})) \\
&= \underset{q(\theta)}{\arg\min}\ \ \mathbb{E}_{q(\theta)}\left[\log \frac{q(\theta|\lambda)}{p(\theta|\mathcal{D})}\right] && \text{definition of KL} \\
&= \underset{q(\theta)}{\arg\min}\ \ \mathbb{E}_{q(\theta)}\left[\log \frac{q(\theta|\lambda)p(\mathcal{D})}{p(\theta,\mathcal{D})}\right] && \text{definition of posterior} \\
&= \underset{q(\theta)}{\arg\min}\ \ \mathbb{E}_{q(\theta)}\left[\log \frac{q(\theta|\lambda)}{p(\theta,\mathcal{D})}\right] + \log p(\mathcal{D}) && \text{constant} \\
&= \underset{q(\theta)}{\arg\min}\ \ -\mathbb{E}_{q(\theta)}\left[\log \frac{p(\theta,\mathcal{D})}{q(\theta|\lambda)}\right] && \text{property of log} \\
&= \underset{q(\theta)}{\arg\max}\ \ \mathbb{E}_{q(\theta)}\left[\log p(\theta,\mathcal{D})\right] + \mathbb{H}(q(\theta)) && \text{ELBO}
\end{aligned}
$$

All quantities are either tractable or easy to estimate by sampling!

You know this from our first class on latent variable models. I'm repeating it here for convenience and with the variables that are relevant for this class.

Note I omit α here and there (due to lack of space), I hope it's not confusing.

The important message: picking the best posterior approximation is exactly equivalent to optimising the ELBO, no approximations there. Note the difference between this and VAEs. In a VAE, we optimise our choice of model as well (we search for θ), and even though we would like (in a VAE) to pick θ that leads to maximum log-likelihood, we can only pick θ that maximises a lowerbound. In VB, we are not searching for θ, as it is given random treatment, rather we search for the optimum of the lowerbound w.r.t. $q(\theta)$. That optimum is exactly the optimum of the VI objective $\arg\min_{q(\theta)}\ \mathrm{KL}(q(\theta)||p(\theta|\mathcal{D}))$.

ELBO continued

Parametric assumption

$$\begin{aligned}&\arg\max_{q(\theta)} \; \mathbb{E}_{q(\theta)}\left[\log p(\theta, \mathcal{D})\right] + \mathbb{H}(q(\theta)) \\ &= \arg\max_{\lambda} \; \mathbb{E}_{q(\theta|\lambda)}\left[\log p(\theta, \mathcal{D})\right] + \mathbb{H}(q(\theta|\lambda))\end{aligned}$$

Recall

$$p(\theta, \mathcal{D}) = p(\theta) \prod_{i=1}^{N} p(y^{(i)}|\theta, x^{(i)})$$

And thus the ELBO evaluates to

$$\begin{aligned}&\mathbb{E}_{q(\theta|\lambda)}\left[\log p(\theta) + \sum_{i=1}^{N} \log p(y^{(i)}|\theta, x^{(i)})\right] + \mathbb{H}(q(\theta|\lambda)) \\ &= \mathbb{E}_{q(\theta|\lambda)}\left[\sum_{i=1}^{N} \log p(y^{(i)}|\theta, x^{(i)})\right] - \mathrm{KL}(q(\theta|\lambda)||p(\theta))\end{aligned}$$

Another difference with VAEs as you know. In VB the latent variable (θ) is global to **all** observations. The latent variable (z) in a VAE is assigned locally per data point.

Mean Field Assumption

Let $\theta \in \mathbb{R}^D$. The simplest approximate posterior is

We can group parameters and assume independence of groups (e.g. layers).

Again, we design posterior approximations with tractability in mind.

Mean Field Assumption

Let $\theta \in \mathbb{R}^D$. The simplest approximate posterior is

$$q(\theta|\lambda) = \prod_{d=1}^{D} q(\theta_d|\lambda)$$

where we assume independence amongst θ_d.

We can group parameters and assume independence of groups (e.g. layers).

Again, we design posterior approximations with tractability in mind.

Mean Field Assumption

Let $\theta \in \mathbb{R}^D$. The simplest approximate posterior is

$$q(\theta|\lambda) = \prod_{d=1}^{D} q(\theta_d|\lambda)$$

where we assume independence amongst θ_d.

If we have the same exponential family for $p(\theta)$ and $q(\theta|\lambda)$,

We can group parameters and assume independence of groups (e.g. layers).

Again, we design posterior approximations with tractability in mind.

Mean Field Assumption

Let $\theta \in \mathbb{R}^D$. The simplest approximate posterior is

$$q(\theta|\lambda) = \prod_{d=1}^{D} q(\theta_d|\lambda)$$

where we assume independence amongst θ_d.

If we have the same exponential family for $p(\theta)$ and $q(\theta|\lambda)$, then

$$\mathrm{KL}(q(\theta|\lambda)||p(\theta)) = \sum_{d=1}^{D} \underbrace{\mathrm{KL}(q(\theta_d|\lambda)||p(\theta_d))}_{\text{closed form}}$$

is known in closed form.

We can group parameters and assume independence of groups (e.g. layers).

Again, we design posterior approximations with tractability in mind.

Choosing λ

How should we choose λ?

Isn't it cool to use a lot of DL machinery to help DL go beyond DL?

Choosing λ

How should we choose λ?

How can we approach the following problem?

$$\arg\max_{\lambda}\ \mathbb{E}_{q(\theta|\lambda)}\left[\log p(\mathcal{D}|\theta)\right] - \underbrace{\mathrm{KL}(q(\theta|\lambda)||p(\theta))}_{\text{closed form}}$$

Isn't it cool to use a lot of DL machinery to help DL go beyond DL?

Choosing λ

How should we choose λ?

How can we approach the following problem?

$$\arg\max_{\lambda} \; \mathbb{E}_{q(\theta|\lambda)}\left[\log p(\mathcal{D}|\theta)\right] - \underbrace{\mathrm{KL}(q(\theta|\lambda)||p(\theta))}_{\text{closed form}}$$

What's the workhorse of optimisation in deep learning?

Isn't it cool to use a lot of DL machinery to help DL go beyond DL?

Gradient-based optimisation for λ

We take steps in the direction that maximises the ELBO

$$\boldsymbol{\nabla}_\lambda \, \mathrm{ELBO} = \boldsymbol{\nabla}_\lambda \mathbb{E}_{q(\theta|\lambda)} \left[\sum_{i=1}^{N} \log p(y^{(i)}|\theta, x^{(i)}) \right] - \boldsymbol{\nabla}_\lambda \, \mathrm{KL}(q(\theta|\lambda)||p(\theta))$$

Gradient-based optimisation for λ

We take steps in the direction that maximises the ELBO

$$\boldsymbol{\nabla}_\lambda \,\mathsf{ELBO} = \boldsymbol{\nabla}_\lambda \mathbb{E}_{q(\theta|\lambda)}\left[\sum_{i=1}^{N} \log p(y^{(i)}|\theta, x^{(i)})\right] - \boldsymbol{\nabla}_\lambda \,\mathsf{KL}(q(\theta|\lambda)||p(\theta))$$

By assumption (mean field and exponential families), KL is tractable (we pack it in a node and autodiff does the job)!

Gradient-based optimisation for λ

We take steps in the direction that maximises the ELBO

$$\nabla_\lambda \, \text{ELBO} = \nabla_\lambda \mathbb{E}_{q(\theta|\lambda)}\left[\sum_{i=1}^{N} \log p(y^{(i)}|\theta, x^{(i)})\right] - \nabla_\lambda \, \text{KL}(q(\theta|\lambda)||p(\theta))$$

By assumption (mean field and exponential families), KL is tractable (we pack it in a node and autodiff does the job)!

How about the first term? What if N is prohibitively large?

Gradient-based optimisation for λ

We take steps in the direction that maximises the ELBO

$$\boldsymbol{\nabla}_\lambda \,\text{ELBO} = \boldsymbol{\nabla}_\lambda \mathbb{E}_{q(\theta|\lambda)}\left[\sum_{i=1}^{N} \log p(y^{(i)}|\theta, x^{(i)})\right] - \boldsymbol{\nabla}_\lambda \,\text{KL}(q(\theta|\lambda)||p(\theta))$$

By assumption (mean field and exponential families), KL is tractable (we pack it in a node and autodiff does the job)!

How about the first term? What if N is prohibitively large?

$$\sum_{i=1}^{N} \log p(y^{(i)}|\theta, x^{(i)}) \quad \text{is certainly prohibitive!}$$

Stochastic gradients are allowed

Noisy, but unbiased, gradients:

$$\nabla_{\lambda}\mathbb{E}_{q(\theta|\lambda)}\left[\sum_{i=1}^{N}\log p(y^{(i)}|\theta, x^{(i)})\right]$$

Data sub-sampling! You know this, but I want to repeat the argument this time for BNNs. Spoiler alert: we will justify mini-batching here.

Stochastic gradients are allowed

Noisy, but unbiased, gradients:

$$\nabla_\lambda \mathbb{E}_{q(\theta|\lambda)}\left[\sum_{i=1}^{N} \log p(y^{(i)}|\theta, x^{(i)})\right]$$

$$= \nabla_\lambda \mathbb{E}_{q(\theta|\lambda)}\left[N \sum_{i=1}^{N} \frac{1}{N} \log p(y^{(i)}|\theta, x^{(i)})\right] \qquad \text{multiply by } N/N$$

Data sub-sampling! You know this, but I want to repeat the argument this time for BNNs. Spoiler alert: we will justify mini-batching here.

Stochastic gradients are allowed

Noisy, but unbiased, gradients:

$$\begin{aligned}
&\nabla_\lambda \mathbb{E}_{q(\theta|\lambda)} \left[\sum_{i=1}^{N} \log p(y^{(i)}|\theta, x^{(i)}) \right] \\
&= \nabla_\lambda \mathbb{E}_{q(\theta|\lambda)} \left[N \sum_{i=1}^{N} \frac{1}{N} \log p(y^{(i)}|\theta, x^{(i)}) \right] \qquad \text{multiply by } N/N \\
&= \nabla_\lambda \mathbb{E}_{q(\theta|\lambda)} \left[N \mathbb{E}_{I \sim \mathcal{U}(1/N)} \left[\log p(y^{(I)}|\theta, x^{(I)}) \right] \right]
\end{aligned}$$

Data sub-sampling! You know this, but I want to repeat the argument this time for BNNs. Spoiler alert: we will justify mini-batching here.

Stochastic gradients are allowed

Noisy, but unbiased, gradients:

$$
\begin{aligned}
&\boldsymbol{\nabla}_\lambda \mathbb{E}_{q(\theta|\lambda)}\left[\sum_{i=1}^{N} \log p(y^{(i)}|\theta, x^{(i)})\right] \\
&= \boldsymbol{\nabla}_\lambda \mathbb{E}_{q(\theta|\lambda)}\left[N \sum_{i=1}^{N} \frac{1}{N} \log p(y^{(i)}|\theta, x^{(i)})\right] && \text{multiply by } N/N \\
&= \boldsymbol{\nabla}_\lambda \mathbb{E}_{q(\theta|\lambda)}\left[N \mathbb{E}_{I\sim\mathcal{U}(1/N)}\left[\log p(y^{(I)}|\theta, x^{(I)})\right]\right] \\
&= N \boldsymbol{\nabla}_\lambda \mathbb{E}_{I\sim\mathcal{U}(1/N)}\left[\mathbb{E}_{q(\theta|\lambda)}\left[\log p(y^{(I)}|\theta, x^{(I)})\right]\right] && \text{swap expectations}
\end{aligned}
$$

Data sub-sampling! You know this, but I want to repeat the argument this time for BNNs. Spoiler alert: we will justify mini-batching here.

Stochastic gradients are allowed

Noisy, but unbiased, gradients:

$$
\begin{aligned}
&\boldsymbol{\nabla}_\lambda \mathbb{E}_{q(\theta|\lambda)}\left[\sum_{i=1}^{N} \log p(y^{(i)}|\theta, x^{(i)})\right] \\
&= \boldsymbol{\nabla}_\lambda \mathbb{E}_{q(\theta|\lambda)}\left[N \sum_{i=1}^{N} \frac{1}{N} \log p(y^{(i)}|\theta, x^{(i)})\right] && \text{multiply by } N/N \\
&= \boldsymbol{\nabla}_\lambda \mathbb{E}_{q(\theta|\lambda)}\left[N \mathbb{E}_{I\sim\mathcal{U}(1/N)}\left[\log p(y^{(I)}|\theta, x^{(I)})\right]\right] \\
&= N \boldsymbol{\nabla}_\lambda \mathbb{E}_{I\sim\mathcal{U}({}^1\!/_N)}\left[\mathbb{E}_{q(\theta|\lambda)}\left[\log p(y^{(I)}|\theta, x^{(I)})\right]\right] && \text{swap expectations} \\
&= N \mathbb{E}_{I\sim\mathcal{U}(1/N)}\left[\boldsymbol{\nabla}_\lambda \mathbb{E}_{q(\theta|\lambda)}\left[\log p(y^{(I)}|\theta, x^{(I)})\right]\right] && \text{linearity}
\end{aligned}
$$

Data sub-sampling! You know this, but I want to repeat the argument this time for BNNs. Spoiler alert: we will justify mini-batching here.

Stochastic gradients are allowed

Noisy, but unbiased, gradients:

$$
\begin{aligned}
&\nabla_\lambda \mathbb{E}_{q(\theta|\lambda)}\left[\sum_{i=1}^{N} \log p(y^{(i)}|\theta, x^{(i)})\right] \\
&= \nabla_\lambda \mathbb{E}_{q(\theta|\lambda)}\left[N \sum_{i=1}^{N} \frac{1}{N} \log p(y^{(i)}|\theta, x^{(i)})\right] && \text{multiply by } N/N \\
&= \nabla_\lambda \mathbb{E}_{q(\theta|\lambda)}\left[N \mathbb{E}_{I\sim\mathcal{U}(1/N)}\left[\log p(y^{(I)}|\theta, x^{(I)})\right]\right] \\
&= N \nabla_\lambda \mathbb{E}_{I\sim\mathcal{U}(1/N)}\left[\mathbb{E}_{q(\theta|\lambda)}\left[\log p(y^{(I)}|\theta, x^{(I)})\right]\right] && \text{swap expectations} \\
&= N \mathbb{E}_{I\sim\mathcal{U}(1/N)}\left[\nabla_\lambda \mathbb{E}_{q(\theta|\lambda)}\left[\log p(y^{(I)}|\theta, x^{(I)})\right]\right] && \text{linearity} \\
&\overset{\text{MC}}{\approx} \frac{N}{M} \sum_{i=1}^{M} \nabla_\lambda \mathbb{E}_{q(\theta|\lambda)}\left[\log p(y^{(i)}|\theta, x^{(i)})\right] && I \sim \mathcal{U}(1/N)
\end{aligned}
$$

Sample a batch, solve expected value under q, then take gradient.

Data sub-sampling! You know this, but I want to repeat the argument this time for BNNs. Spoiler alert: we will justify mini-batching here.

Challenge

$$\boldsymbol{\nabla}_\lambda \,\text{ELBO} =$$

$$\boldsymbol{\nabla}_\lambda \mathbb{E}_{q(\theta|\lambda)}\left[\sum_{i=1}^{N} \log p(y^{(i)}|\theta, x^{(i)})\right] - \boldsymbol{\nabla}_\lambda \,\text{KL}(q(\theta|\lambda)||p(\theta))$$

It gets better every time, but we still have to differentiate an intractable expected value. If we only knew a technique for stochastic backpropagation :-)

Challenge

$$\nabla_\lambda \text{ELBO} =$$

$$\nabla_\lambda \mathbb{E}_{q(\theta|\lambda)}\left[\sum_{i=1}^{N} \log p(y^{(i)}|\theta, x^{(i)})\right] - \nabla_\lambda \text{KL}(q(\theta|\lambda)||p(\theta))$$

$$= N\mathbb{E}_{\mathcal{U}(1/N)}\left[\nabla_\lambda \mathbb{E}_{q(\theta|\lambda)}\left[\log p(y^{(I)}|\theta, x^{(I)})\right]\right] - \nabla_\lambda \text{KL}(q(\theta|\lambda)||p(\theta))$$

It gets better every time, but we still have to differentiate an intractable expected value. If we only knew a technique for stochastic backpropagation :-)

Challenge

$\boldsymbol{\nabla}_\lambda \, \text{ELBO} =$

$$\boldsymbol{\nabla}_\lambda \mathbb{E}_{q(\theta|\lambda)}\left[\sum_{i=1}^{N} \log p(y^{(i)}|\theta, x^{(i)})\right] - \boldsymbol{\nabla}_\lambda \, \text{KL}(q(\theta|\lambda)||p(\theta))$$

$$= N\mathbb{E}_{\mathcal{U}(1/N)}\left[\boldsymbol{\nabla}_\lambda \mathbb{E}_{q(\theta|\lambda)}\left[\log p(y^{(l)}|\theta, x^{(l)})\right]\right] - \boldsymbol{\nabla}_\lambda \, \text{KL}(q(\theta|\lambda)||p(\theta))$$

- we can compute KL and thus differentiate it
- mini-batching is allowed, so we can compute the first term for a few datapoints at a time
- but can we really solve $\mathbb{E}_{q(\theta|\lambda)}\left[\log p(y^{(i)}|\theta, x^{(i)})\right]$ for even a single instance?

It gets better every time, but we still have to differentiate an intractable expected value. If we only knew a technique for stochastic backpropagation :-)

Reparameterisation

Remember the *law of the unconscious statistician*?

$$\mathbb{E}_{q(\theta|\lambda)}\left[f(\theta)\right] = \mathbb{E}_{\phi(\epsilon)}\left[f(\theta = \mathcal{T}^{-1}(\epsilon, \lambda))\right]$$

We used it for VAEs, and we are going to use it now for BNNs.

Parameters are almost always continuous, so reparameterised gradients should not be too difficult here.

Reparameterised Gradients

Assume we pick $q(\theta|\lambda)$ from a reparameterisable family
e.g. location-scale distributions

The trick here is to show that data sub-sampling and reparameterisation can be swapped around.

And remember, if we don't know KL exactly, or if there are dependencies across subsets of θ, we can always use a reparameterised gradient for it, since KL is an expected value under $q(\theta|\lambda)$.

Reparameterised Gradients

Assume we pick $q(\theta|\lambda)$ from a reparameterisable family
e.g. location-scale distributions

$\nabla_\lambda \text{ELBO} =$

$$N\mathbb{E}_I\left[\nabla_\lambda \mathbb{E}_{q(\theta|\lambda)}\left[\log p(y^{(I)}|\theta, x^{(I)})\right]\right] - \nabla_\lambda \text{KL}(q(\theta|\lambda)||p(\theta))$$

The trick here is to show that data sub-sampling and reparameterisation can be swapped around.

And remember, if we don't know KL exactly, or if there are dependencies across subsets of θ, we can always use a reparameterised gradient for it, since KL is an expected value under $q(\theta|\lambda)$.

Reparameterised Gradients

Assume we pick $q(\theta|\lambda)$ from a reparameterisable family
e.g. location-scale distributions

$$\nabla_\lambda \, \mathrm{ELBO} =$$

$$N\mathbb{E}_I\left[\nabla_\lambda \mathbb{E}_{q(\theta|\lambda)}\left[\log p(y^{(I)}|\theta, x^{(I)})\right]\right] - \nabla_\lambda \, \mathrm{KL}(q(\theta|\lambda)||p(\theta))$$

$$= N\mathbb{E}_I\left[\nabla_\lambda \mathbb{E}_{\underline{\phi(\epsilon)}}\left[\log p(y^{(I)}|\underline{\theta = \mathcal{T}^{-1}(\epsilon, \lambda)}, x^{(I)})\right]\right] - \nabla_\lambda \, \mathrm{KL}(q(\theta|\lambda)||p(\theta))$$

The trick here is to show that data sub-sampling and reparameterisation can be swapped around.

And remember, if we don't know KL exactly, or if there are dependencies across subsets of θ, we can always use a reparameterised gradient for it, since KL is an expected value under $q(\theta|\lambda)$.

Reparameterised Gradients

Assume we pick $q(\theta|\lambda)$ from a reparameterisable family
e.g. location-scale distributions

$$\boldsymbol{\nabla}_\lambda \,\mathrm{ELBO} =$$

$$N\mathbb{E}_I\left[\boldsymbol{\nabla}_\lambda \mathbb{E}_{q(\theta|\lambda)}\left[\log p(y^{(I)}|\theta, x^{(I)})\right]\right] - \boldsymbol{\nabla}_\lambda \,\mathrm{KL}(q(\theta|\lambda)||p(\theta))$$

$$= N\mathbb{E}_I\left[\boldsymbol{\nabla}_\lambda \mathbb{E}_{\underline{\phi(\epsilon)}}\left[\log p(y^{(I)}|\underline{\theta = \mathcal{T}^{-1}(\epsilon, \lambda)}, x^{(I)})\right]\right] - \boldsymbol{\nabla}_\lambda \,\mathrm{KL}(q(\theta|\lambda)||p(\theta))$$

$$= N\mathbb{E}_I\left[\mathbb{E}_{\phi(\epsilon)}\left[\boldsymbol{\nabla}_\lambda \log p(y^{(I)}|\mathcal{T}^{-1}(\epsilon, \lambda), x^{(I)})\right]\right] - \boldsymbol{\nabla}_\lambda \,\mathrm{KL}(q(\theta|\lambda)||p(\theta))$$

The trick here is to show that data sub-sampling and reparameterisation can be swapped around.

And remember, if we don't know KL exactly, or if there are dependencies across subsets of θ, we can always use a reparameterised gradient for it, since KL is an expected value under $q(\theta|\lambda)$.

Reparameterised Gradients

Assume we pick $q(\theta|\lambda)$ from a reparameterisable family
e.g. location-scale distributions

$$\nabla_\lambda \,\mathrm{ELBO} =$$

$$N\mathbb{E}_I\left[\nabla_\lambda \mathbb{E}_{q(\theta|\lambda)}\left[\log p(y^{(I)}|\theta, x^{(I)})\right]\right] - \nabla_\lambda \,\mathrm{KL}(q(\theta|\lambda)||p(\theta))$$

$$= N\mathbb{E}_I\left[\nabla_\lambda \mathbb{E}_{\underline{\phi(\epsilon)}}\left[\log p(y^{(I)}|\underline{\theta = \mathcal{T}^{-1}(\epsilon, \lambda),} x^{(I)})\right]\right] - \nabla_\lambda \,\mathrm{KL}(q(\theta|\lambda)||p(\theta))$$

$$= N\mathbb{E}_I\left[\mathbb{E}_{\phi(\epsilon)}\left[\nabla_\lambda \log p(y^{(I)}|\mathcal{T}^{-1}(\epsilon, \lambda), x^{(I)})\right]\right] - \nabla_\lambda \,\mathrm{KL}(q(\theta|\lambda)||p(\theta))$$

$$= N\mathbb{E}_{\phi(\epsilon)}\left[\mathbb{E}_I\left[\nabla_\lambda \log p(y^{(I)}|\mathcal{T}^{-1}(\epsilon, \lambda), x^{(I)})\right]\right] - \nabla_\lambda \,\mathrm{KL}(q(\theta|\lambda)||p(\theta))$$

The trick here is to show that data sub-sampling and reparameterisation can be swapped around.

And remember, if we don't know KL exactly, or if there are dependencies across subsets of θ, we can always use a reparameterised gradient for it, since KL is an expected value under $q(\theta|\lambda)$.

Reparameterised Gradient Estimate

$$\nabla_\lambda \text{ELBO} =$$

$$N\mathbb{E}_{\phi(\epsilon)}\left[\mathbb{E}_I\left[\nabla_\lambda \log p(y^{(I)}|\mathcal{T}^{-1}(\epsilon,\lambda), x^{(I)})\right]\right] - \nabla_\lambda \text{KL}(q(\theta|\lambda)||p(\theta))$$

$$\overset{\text{MC}}{\approx} \left(\frac{M}{NK}\sum_{k=1}^{K}\sum_{i=1}^{M} \nabla_\lambda \log p(y^{(i)}|\underbrace{\mathcal{T}^{-1}(\epsilon^{(k)},\lambda)}_{=\theta^{(k)}}, x^{(i)})\right)$$

$$- \nabla_\lambda \text{KL}(q(\theta|\lambda)||p(\theta))$$

where $\epsilon^{(k)} \sim \phi(\epsilon)$, $\theta^{(k)} = \mathcal{T}^{-1}(\epsilon^{(k)}, \lambda) \sim q(\theta|\lambda)$, and $I \sim \mathcal{U}(1/N)$

Procedure

- Sample parameters via deterministic reparameterisation
- Sample batch
- Compute likelihood and KL: forward
- Sampling parameters first allows for efficient parallel implementation

Once I told you that scaling the gradient of the log-likelihood term would matter in the future. Here it is. It matters because data sub-sampling affects only the log-likelihood part, not the KL part.

Interpret this, we are doing mini-batch training, where each batch uses a different set of parameters, the parameters are not being optimised, but they are sampled from an approximation to the model's true posterior, this approximation is being optimised. Once again: there are no ∇_θ terms, whatsoever. Do you see that?

After training?

After training, we don't have 1 model, we have a distribution $q(\theta|\lambda)$ over "all possible models"

- $q(\theta|\mathcal{D})$ approximates the true posterior $p(\theta|\mathcal{D})$
- it should prefer models that are likely after observing data $\mathcal{D}$ in light of whatever prior assumptions we made
- there are no convergence guarantees and most approximating families are too simple (underestimate variance)

Training now gives you a point estimate for λ
so we are not training $p(\mathcal{D}|\theta)$, we are training $q(\theta|\lambda)$!

After training?

After training, we don't have 1 model, we have a distribution $q(\theta|\lambda)$ over "all possible models"

- $q(\theta|\mathcal{D})$ approximates the true posterior $p(\theta|\mathcal{D})$
- it should prefer models that are likely after observing data $\mathcal{D}$ in light of whatever prior assumptions we made
- there are no convergence guarantees and most approximating families are too simple (underestimate variance)

After training we make inferences using $q(\theta|\lambda)$

- $p(y_*|\mathcal{D}, x_*) \approx \int q(\theta|\lambda)p(y_*|\theta, x_*)\mathrm{d}\theta$
  which we typically further approximate via sampling

Training now gives you a point estimate for λ
so we are not training $p(\mathcal{D}|\theta)$, we are training $q(\theta|\lambda)$!

After training?

After training, we don't have 1 model, we have a distribution $q(\theta|\lambda)$ over "all possible models"

- $q(\theta|\mathcal{D})$ approximates the true posterior $p(\theta|\mathcal{D})$
- it should prefer models that are likely after observing data $\mathcal{D}$ in light of whatever prior assumptions we made
- there are no convergence guarantees and most approximating families are too simple (underestimate variance)

After training we make inferences using $q(\theta|\lambda)$

- $p(y_*|\mathcal{D}, x_*) \approx \int q(\theta|\lambda)p(y_*|\theta, x_*)\mathrm{d}\theta$
  which we typically further approximate via sampling

We can also estimate $p(\mathcal{D})$ using q and the importance sampling fundamental identity, i.e. $p(\mathcal{D}) = \int q(\theta|\lambda)\frac{p(\theta,\mathcal{D})}{q(\theta|\lambda)}\mathrm{d}\theta$

Training now gives you a point estimate for λ
so we are not training $p(\mathcal{D}|\theta)$, we are training $q(\theta|\lambda)$!

Terminology

Approximate posterior $q(\theta|\lambda)$

Variational inference $\arg\min_{q(\theta)} \; \mathrm{KL}(q(\theta)||p(\theta|\mathcal{D}))$

ELBO $\mathbb{E}_{q(\theta)}[\log p(\mathcal{D}|\theta)] - \mathrm{KL}(q(\theta)||p(\theta))$

Mean field assumption $q(\theta|\lambda) = \prod_{d=1}^{D} q(\theta_d|\lambda_d)$

Reparameterised gradients

$$\nabla_\lambda \mathbb{E}_{q(\theta|\lambda)}\left[f(\theta)\right] = \mathbb{E}_{\phi(\epsilon)}\left[\nabla_\theta f(\theta) \nabla_\lambda t(\epsilon, \lambda)\right]$$

Posterior predictive distribution

$$p(y_*|\mathcal{D}, x_*) \approx \int q(\theta|\lambda) p(y_*|\theta, x_*) \mathrm{d}\theta$$

Summary

VI tuns inference into optimisation and gives you a proxy to $p(\theta|\mathcal{D})$

Estimates of posterior predictive mean and variance

- help you decide whether or not to make a decision
    - in classification: consider plotting precision and recall against predictive variance
    - in regression: interval in which you expect a response to be

Estimates of marginal likelihood

- help you compare models under different hyperparameters

Caveat: limited understanding about the impact of our priors

Literature

Variational inference	Blei et al. (2017)
Stochastic VI	Hoffman et al. (2013)
for nonconjugate inference	Titsias and Lázaro-Gredilla (2014)
Bayes by backprop	Blundell et al. (2015)
Model comparison	MacKay (1992a)

Outline

1. Bayes: what and why?
2. Choosing a prior
3. Posterior Inference for BNNs
4. **Bayesian Dropout**
5. Example

Dropout

A **very** simple technique to make MLE more robust

- stochastic training: with probability $1 - p$, “drop” inputs to a fully connected layer
- possibly use L_2 regularisation (because why not?)
- deterministic test: disable “dropout” and scale weights by p

Srivastava et al. (2014)

Dropout is a much loved ‘regularisation’ scheme for NNs.

Every NN that is worth anything probably is trained with some dropout.

Relate dropout to BNNs

BNNs come with a somewhat disappointing fact, that we have no clue what classes of random functions a given prior leads to.

- BNNs however can be seen as an approximation to a GP: there are connections between the nonlinearities we use and known kernels

Gal and Ghahramani (2016b) show that a VB procedure for a certain BNN is an approximate inference scheme for an approximation to a GP. Moreover, with a specific choice of parametric family for $q(\theta|\lambda)$, this VB procedure is identical to MLE-training with dropout (up to some additional ‘regularisers’).

- The main consequences take place after trainingL we gain access to estimates of marginal likelihood and posterior predictive distribution.

About priors for BNNs

- To understand more about the role of priors and Bayesian averaging in BNNs, check Wilson (2020).

About approximate inference for an approximate GP:

- A BNN is a parametric approximation to a GP (MacKay, 1992b; Neal, 1994; Gal and Ghahramani, 2016b).
- MLE-training with dropout comes very close to a VB algorithm for BNNs. Thus training with dropout can be thought as approximate inference for a BNN (and thus for an approximation to a GP).
- The connection to GPs is interesting from a theoretical standpoint, but might not tell you much.
- The practical implication, however, is that a tractable VB algorithm allows us to estimate the result of posterior predictive queries for BNNs.

Single hidden layer BNN

Consider this simple Bayesian FFNN with a single hidden layer

$$\begin{aligned} \mathbf{h} &= \sigma(\mathbf{W}_1 x + \mathbf{b}) \\ f(x) &= \mathbf{W}_2 \mathbf{h} \end{aligned}$$

where

- $x \in \mathbb{R}^D$ is an input (predictor), and $f(x) \in \mathbb{R}^O$ is a latent output
- $\mathbf{W}_1 \in \mathbb{R}^{H \times D}, \mathbf{b} \in \mathbb{R}^H, \mathbf{W}_2 \in \mathbb{R}^{O \times H}$
  - each of the D columns of $\mathbf{W}_1$ is distributed by $\mathcal{N}(0, \ell^{-2}\mathbb{I}_H)$
  - $\mathbf{b} \sim \mathcal{N}(0, \ell_0^{-2}\mathbb{I}_H)$
  - each of the H columns of $\mathbf{W}_2$ is distributed by $\mathcal{N}(0, \ell^{-2}\mathbb{I}_O)$
- $\sigma(\cdot)$ is an elementwise non-linearity (e.g., sigmoid, softplus, tanh).

This is a FFNN, but it is a Bayesian one, because its parameters are rvs.

The latent output is used to parameterise a likelihood, e.g.
$Y|x, \tau \sim \mathcal{N}(f(x;\theta), \tau^{-1}\mathbb{I}_O)$ in regression or
$Y|x \sim \mathrm{Cat}(\mathrm{softmax}(f(x;\theta)))$ in classification.

Parametric approximation to GP: it turns out this simple block can be thought of as a parametric approximation to a GP. As the number of hidden units H goes to infinity, we recover the non-parametric GP (MacKay, 1992b; Neal, 1994; Gal and Ghahramani, 2016b).

VB for Bayesian FFNN

We search for a parametric approximation to the model's true posterior distribution that minimises $\mathsf{KL}\left(q(\theta|\lambda) \mid\mid p(\theta|\mathcal{D})\right)$. For that, we can optimise the ELBO w.r.t. λ:

$$\arg\max_{\lambda} \ \mathbb{E}_{q(\theta|\lambda)}[\log p(\mathcal{D}|\theta)] - \mathsf{KL}\left(q(\theta|\lambda) \mid\mid p(\theta)\right)$$

This takes specifying $q(\theta)$ where $\theta = \{\mathbf{W}_1, \mathbf{b}, \mathbf{W}_2\}$.

In the next few slides, we will derive the VB procedure that recovers dropout. Before going ahead with that, can you design a VB procedure that you find convenient? Don't worry about making it look like dropout at all, use the knowledge of VB you have gathered so far.

A mean field approximation that recovers dropout

Independence across parameter groups

$$q(\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}|\lambda) = \left(\prod_{c=1}^{D} q(\mathbf{w}_{1,c}|\lambda_{1,c})\right) \left(\prod_{c=1}^{H} q(\mathbf{w}_{2,c}|\lambda_{2,c})\right) q(\mathbf{b}|\lambda_b)$$

Diagonal Gaussian for biases

$$q(\mathbf{b}|\lambda) = \mathcal{N}(\mathbf{b}|\mathbf{m}, \sigma^2\mathbb{I}_H) \quad \text{nicely reparameterisable!}$$

Mixture of Gaussians for each row of $\mathbf{W}_1$ or $\mathbf{W}_2$

$$q(\mathbf{w}_{1,c}|\lambda_1) = p\mathcal{N}(\mathbf{w}_{1,c}|\mathbf{m}_{1,c}, \sigma^2\mathbb{I}_H) + (1-p)\mathcal{N}(\mathbf{w}_{1,c}|\mathbf{0}, \sigma^2\mathbb{I}_H)$$
$$q(\mathbf{w}_{2,c}|\lambda_2) = p\mathcal{N}(\mathbf{w}_{2,c}|\mathbf{m}_{2,c}, \sigma^2\mathbb{I}_O) + (1-p)\mathcal{N}(\mathbf{w}_{2,c}|\mathbf{0}, \sigma^2\mathbb{I}_O)$$

We are going to make a mean field assumption and pick convenient families for each factor.

For biases, we get a diagonal Gaussian.

For columns of the linear layers we will mix two Gaussians using $p \in (0, 1)$. For one of the Gaussians we learn a mean vector, for the other we used $\mathbf{0}$.

In all cases, assume the variances are fixed, as well as p. Thus $\lambda = \{\mathbf{m}, \mathbf{M}_1, \mathbf{M}_2\}$ where $\mathbf{M}_1$ is the $H \times D$ matrix obtained by concatenating the (column) vectors $\mathbf{m}_{1,c}$ for $c = 1, \ldots, D$, and $\mathbf{M}_2$ is the $O \times H$ matrix obtained by concatenating the (column) vectors $\mathbf{m}_{2,c}$ for $c = 1, \ldots, H$.

Can we reparameterise samples from the mixture of Gaussians?

Mixture of Deltas

Let $\sigma \to 0$

- from mixture of Gaussians to mixture of Deltas

$$Z|p \sim \text{Bern}(p)$$

$$\mathbf{w}_{1,c} = \begin{cases} \mathbf{m}_{1,c} + \mathbf{0} \odot \epsilon & \text{if } z_{1,c} = 1 \\ (1 - z_{1,c})\mathbf{0} \odot \epsilon & \text{if } z_{1,c} = 0 \end{cases}$$

$$= z_{1,c}\mathbf{m}_{1,c}$$

- biases are deterministic because $\sigma \to 0$ the Gaussian tends to $\delta(\mathbf{m} - \mathbf{b})$

The affine transform in fully connected layers becomes

$$([(z_{1,c}\mathbf{m}_{1,c})]_{c=1}^{D})x + \mathbf{m} = ([\mathbf{m}_{1,c}]_{c=1}^{D})(\mathbf{z}_1 \odot x) + \mathbf{m}$$

with probability p, we essentially drop inputs

Same happens with $\mathbf{W}_2$ (weights of the second layer)

Here Gal and Ghahramani (2016b) make an unusual assumption, namely, that the variational Gaussians have **no variance**. This is the kind of thing you only do when you are really trying to get to a procedure that really looks like dropout. There is merit in this (it recovers dropout), but there are problems as well. Can you anticipate some problems?

Notation

- $[(z_{1,c}\mathbf{m}_{1,c})]_{c=1}^{D}$ makes an $H \times D$ matrix by concatenating (column) vectors in sequence;
- $\mathbf{z}_1$ is a vector of D independent Bernoulli draws

Note how we can mask the columns using the Bernoulli samples, or alternatively, mask the inputs. Do you see the advantage of being able to push the randomness to the input?

ELBO

We can now use stochastic backpropagation to solve

$$\arg\max_{\lambda} \; N\,\mathbb{E}_I\left[\mathbb{E}_Z\left[\log p(y^{(I)}|x^{(I)}, \theta = \mathcal{T}^{-1}(z, \lambda))\right]\right] - \mathrm{KL}(q(\theta|\lambda)||p(\theta))$$

where

- $\theta = \{\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}\}$ and $\lambda = \{\mathbf{M}_1, \mathbf{M}_2, \mathbf{m}\}$
- $Z_{1,i} \sim \mathrm{Bern}(p)$ and $Z_{2,s} \sim \mathrm{Bern}(p)$

We can sample with a reparameterisation

- draws from $\mathrm{Bern}(p)$ are used to mask the inputs to layers

We can assess the likelihood

- because we were the ones to choose it

We are only missing a KL term: which in this case can be approximated by L_2 on λ.

The KL term requires approximation because we have an MoG approximate posterior for $\mathbf{W}_1, \mathbf{W}_2$ and because of the assumption that $\sigma \to 0$.

If you propose a VB procedure yourself, one that's not heavily inspired by dropout, you can choose $q(\theta|\lambda)$ such that the KL term is known in closed-form.

It's always worth remarking: we are optimising $q(\theta|\lambda)$, not $p(\mathcal{D}|\theta)$.

KL approximation

In regression (recall τ is the prior precision for the likelihood $Y|f$)

$$\frac{p}{2\tau N}||\mathbf{M}_1||_2^2 + \frac{p}{2\tau N}||\mathbf{M}_2||_2^2 + \frac{1}{2\tau N}||\mathbf{m}||_2^2 \tag{1}$$

In classification

$$\frac{p}{2N}||\mathbf{M}_1||_2^2 + \frac{p}{2N}||\mathbf{M}_2||_2^2 + \frac{1}{2N}||\mathbf{m}||_2^2 \tag{2}$$

See Gal and Ghahramani (2016b)

Prior parameters

The prior length-scale is the inverse of the standard deviation of the distribution over the scaling weights in affine layers, it controls the rate of change of the sampled functions.

The prior precision (in regression) controls the observation noise, smaller precision leads to bigger error bars

In classification we let $\tau^{-1} \to 0$.

These are hyperparameters you have to search for. You can also relate them to the weight decay if your NN library offers weight decay out of the box.

Approximate posterior parameters

$\mathbf{M}_1$ variational mean for input-to-hidden, $\mathbf{m}$ variational mean of bias vector, $\mathbf{M}_2$ variational mean for hidden-to-output

- these are the only trainable parameters

In principle, we can have one Bernoulli parameter per layer.

Do you think we could learn the Bernoulli parameter easily?

Inferences

Marginal likelihood: get estimates via importance sampling to compare different hyperparameters

Inferences

Marginal likelihood: get estimates via importance sampling to compare different hyperparameters

Posterior predictive distribution

- do not disable dropout at test time

Inferences

Marginal likelihood: get estimates via importance sampling to compare different hyperparameters

Posterior predictive distribution

- do not disable dropout at test time
- compute empirical estimates of predictive mean and variance from stochastic outputs

Inferences

Marginal likelihood: get estimates via importance sampling to compare different hyperparameters

Posterior predictive distribution

- do not disable dropout at test time
- compute empirical estimates of predictive mean and variance from stochastic outputs
- for classification, consider uncertainty over class probabilities, for example, boxplot samples of the probability $p(y|x_*, \theta)$ of each outcome y for $\theta \sim q(\theta|\lambda)$

Inferences

Marginal likelihood: get estimates via importance sampling to compare different hyperparameters

Posterior predictive distribution

- do not disable dropout at test time
- compute empirical estimates of predictive mean and variance from stochastic outputs
- for classification, consider uncertainty over class probabilities, for example, boxplot samples of the probability $p(y|x_*, \theta)$ of each outcome y for $\theta \sim q(\theta|\lambda)$
- multiple forward passes, but a single trained model

Inferences

Marginal likelihood: get estimates via importance sampling to compare different hyperparameters

Posterior predictive distribution

- do not disable dropout at test time
- compute empirical estimates of predictive mean and variance from stochastic outputs
- for classification, consider uncertainty over class probabilities, for example, boxplot samples of the probability $p(y|x_*, \theta)$ of each outcome y for $\theta \sim q(\theta|\lambda)$
- multiple forward passes, but a single trained model
- unlike in an ensemble, we are sampling from $q(\theta|\lambda)$, an approximation to $p(\theta|\mathcal{D})$

Extensions

Note a few things

- to recover dropout, it was crucial to use a mixture of *deltas* as variational approximation
- this allowed us to sample parameters by having a mask over inputs (rather than over parameters)
- this trick seems general, but it does depend on the type of layer we deal with

An RNN is just a FFNN dynamically unfolded through time

- all we need is to sample the mask once per data point
- and reuse the same mask for all steps in the sequence

See Gal and Ghahramani (2016c).

CNNs can be reformulated as a linear operation followed by a pooling non-linearity, in this view we need to drop outputs of the linear operation (a parameterised inner product) before pooling. See Gal and Ghahramani (2016a).

How about Transformer layers?

Summary

Bayesian posterior inference for NNs with negligible training effort

Bayesian posterior predictive at linear cost (one forward pass per sample)

Future research

- better posterior approximations (fewer independence assumptions)
- better handle on properties of kernels
- BNNs typically underestimate variance

Literature

Yarin Gal's thesis and blogpost Gal (2016)

Dropout as Bayesian approximation Gal and Ghahramani (2016b, esp appendix)

CNN and RNN variants Gal and Ghahramani (2016a,c)

Modern take on Bayesian deep learning Wilson (2020); Wilson and Izmailov (2020)

Outline

1. Bayes: what and why?
2. Choosing a prior
3. Posterior Inference for BNNs
4. Bayesian Dropout
5. Example

Detecting hate speech

We have N training data points labelled for hate speech, and we design three models.

MLE

$$Y^{(i)}|\theta, x^{(i)} \sim \text{Bern}(\text{sigmoid}(f(x^{(i)};\theta)))$$

GP

$$\begin{aligned} F^{(1)}, \dots, F^{(N)}|\alpha, x^{(1)}, \dots, x^{(N)} &\sim \mathcal{N}(\mathbf{0}, \mathbf{K}) \\ Y^{(i)}|F^{(i)} &\sim \text{Bern}(\text{sigmoid}(F^{(i)})) \end{aligned}$$

BNN

$$\begin{aligned} \Theta|\alpha &\sim N(0, I_D) \\ Y^{(i)}|\theta, x^{(i)} &\sim \text{Bern}(\text{sigmoid}(f(x^{(i)};\theta))) \end{aligned}$$

Here $K_{i,k} = k(e(x^{(i)}), e(x^{(j)}); \alpha)$ for a choice of kernel, such as the radial basis function, and $e(\cdot)$ an embedding function that maps x to $\mathbb{R}^D$, such as average of fixed and pre-trained word2vec embeddings.

Let's make it tricky!

Suppose the word 'red' can be used to express hateful speech, and suppose that in the training set 30% of the data points that contains 'red' are indeed hateful.

This is a very simple pattern to grasp. A BoW model can learn that.

The NN model is not trained to match the conditional frequency of $Y|\text{'red'} \in x$, rather the conditional frequency of $Y|x$, and the more flexible $f(\cdot;\theta)$ is, the more peaked the predicted likelihood $Y|\theta, x$ can be (that is, $\text{Var}(Y|\theta, x)$ is very low).

A model can explain the pattern about 'red' by learning to identify its meaning in context (like humans attempt to do), or by correlating the label with anything else that seems predictive enough of the pattern.

Assume, for example, that in this dataset it just so happens that every case in which 'red' was hateful the speaker was British. There are plenty of low-level distributional features in text that are predictive of British English. Clearly, none of those has anything to do with 'red' being used to convey hateful speech.

If a BoW model learns the pattern $p(y = \text{hateful}|\text{'red'} \in x, \theta) = 0.3$ does it mean it can classify the precise instances of x where 'red' is used to express hate?

What do you think the conditional observed frequency of $Y = 1|x$ for a fixed x generally is?

I, for example, tend to use British spelling (and that's no more than an accident, for part of my education was in the UK).

How do we know what the model knows?

The NN model has one objective, and one objective alone, to assign the highest likelihood within the reach of its parametric family. It does not need to converge to plausible explanations, it needs to converge to an explanation that makes the data likely.

Though, presumably, there are many such explanations, after all, there are so many ways to correlate that single instance of ‘red’ to the label without grasping what makes ‘red’ such a negative word in any one case.

Suppose we have two test sets: one is held-out from training, in there roughly 30% of the occurrences of ‘red’ are indeed hateful; another is skewed, all of the instances were written by American speakers, and 50% of the instances are hateful.

High $\mathrm{Var}(Y|x_*, \mathcal{D})$ is indicative that there is not enough coherence across the many hypotheses (classifiers) that are supported by all of our observations.

The NN model will always assign a single deterministic probability of hateful speech, no matter what instance of x containing ‘red’ we give it. Weird things that can happen

- the model will assign higher probability for ‘hateful’ for almost all instances containing British spelling in test set 1 (the model has the wrong explanation);
- the model assigns higher probability for ‘hateful’ in about 30% of the cases that contain ‘red’ in test set 1 though these are not the 30% that are indeed hateful (the model got the BoW pattern);
- the model consistently assigns low probability for ‘hateful’ in test set 2 (here it might be overusing the correlation with British spelling)
- the model assigns higher probability for ‘hateful’ in about 30% of the cases in test set 2 (again, the model got the BoW pattern)

We have no clue when we can use it. We just use the model because ‘well, it’s accuracy in test set 1 is 90%’. A number that says so little. What are the 10% of mistakes? Will the mistakes differ if we have other hypotheses supported by the data? What happens in test sets that are not, strictly speaking, in-domain (held-out from training)?

Will it work though?

What do you mean? Are we going to improve the accuracy of decisions?

Accuracy, as you are probably thinking of, has to do with binary assessments of decisions that you are **forced** to make. If you are walking the land of uncertainty estimates, you are past the idea of making decisions blindly. You want to find **support** for your decisions, thus you should reserve yourself the right to say *I cannot decide, consult a human expert*. So you want to look into more than accuracy.

In general, no matter the problem, making a decision can incur loss. You can take the loss into account. For example, perhaps misclassifying normal speech as hateful costs you on average EUR 2 in annotation effort. On the other hand, misclassifying hateful speech as normal can cost you much more in lawsuits (plus extremely negative consequences to the users of your platform, many of which we struggle to even assess).

Rational decision making under uncertainty:

$$\arg\min_{y} \mathbb{E}[\ell(y, Y*)|x*, \mathcal{D}]$$

The expectation is taken w.r.t. the posterior predictive distribution $Y * |x*, \mathcal{D}$, here ℓ assesses the loss of predicting y when the correct label is $y*$ with probability $p(y * |x*, \mathcal{D})$.

See Barber (Chapter 7, 2012) which is freely available online http://web4.cs.ucl.ac.uk/staff/D.Barber/pmwiki/pmwiki.php?n=Brml.HomePage

Ideas for analysing models and supporting decisions

- MC estimates of posterior predictive variance.
- In classification, ditch majority voting, use posterior predictive samples to estimate a confusion matrix.
- In regression, ditch MSE, estimate the probability of the outcome falling in a reasonable interval (e.g., do you really care about predicting 3.2567 stars or whether stars are more likely to be in $(2, 3)$ than in any other unit sub-set?).
- When comparing different models capable of producing uncertainty estimates you can fix a level of certainty (a maximum posterior predictive variance you shall tolerate) and refrain from deciding sometimes, then you can draw precision/recall curves to compare models;
- If you have a sensitive problem, misclassification for certain labels is probably far more dangerous than for others, introduce a loss function and play with it. Compare different models in terms of the decision-making loss.

More ideas:

- Can we correlate classification error and estimates of posterior predictive variance? This can tell us something about the quality of the uncertainty estimate. Ideally, high variance should indicate that decisions are not safe and are likely to lead to errors.
- In regression, how about estimating the smallest interval that will contain the correct answer with X% probability.

Strategy: gather samples from $Y_*|x_*, \mathcal{D}$ and analyse what you got. Use plots, ordered statistics, moments.

Don't embrace Bayesian methods to push the leaderboard, that's unlikely to work. If you are thinking Bayesian you are already thinking deeper than most leaderboards were designed for.

Above all, be critical

Ovadia et al. (2019) present strategies to evaluate uncertainty estimates. Being critical about uncertainty estimates is crucial. Going from NNs to BNNs we enable uncertainty estimates, but the problem of supporting meaningful conclusions is not solved. As variance is something we have to postulate, the way we postulate it and the degree to which we do so matters. There are many knobs and their impact on the quality of uncertainty estimates is not at all obvious.

If we think about our hateful speech detection example, it is true that the BNN model holds potential, but there are clear risks: e.g. Bayesian dropout can get dangerously close to MLE (can you see why?). If that happens we are back to over-confident predictions. Replacing NNs by BNNs and not caring about what is going on beyond reporting held-out accuracy is just as meaningful as ignoring the entire problem all along and go on with MLE forever.

By the way, I would not be surprised if the GP model would be the one with the better tradeoff.

Our Bayesian models all have parameters that affect the conclusions we can draw from them. If in our analysis, we find that we cannot use posterior predictive variance to anticipate bad decisions, that is, to warn against lack of knowledge, then we need to revisit our assumptions (this includes the choice of kernel in a GP, choice of prior and prior hyperaparameters in a BNN). We can revisit our choices in order to improve the meaningfulness of our uncertainty estimates (i.e., to make them more predictive of errors), that is absolutely safe, and is the beauty of the Bayesian paradigm: we can incorporate knowledge into the prior. What we shouldn't not fall victim to is to exploit superficially obvious shortcuts: for example, optimising the choice of prior parameters or choice of kernel on marginal likelihood (evidence) of observations is unlikely to make our posterior predictive estimates any more meaningful. Clearly, this is ML, we can do whatever we like, and we can do that as well, as long as we know the limitations of our tools for data analysis and are transparent about it.

References I

David Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, Cambridge, 2012. ISBN 978-0-521-51814-7. doi: 10.1017/CBO9780511804779. URL https://www.cambridge.org/core/books/bayesian-reasoning-and-machine-learning/37DAFA214EEE41064543384033D2ECF0.

José M Bernardo and Adrian FM Smith. *Bayesian theory*, volume 405. John Wiley & Sons, 2009.

Christopher M. Bishop. *Pattern recognition and machine learning (information science and statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0-387-31073-8. tex.date-added: 2019-09-19 08:15:14 +0000 tex.date-modified: 2019-09-19 08:15:14 +0000.

References II

David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017. Publisher: Taylor & Francis.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In Francis Bach and David Blei, editors, *Proceedings of the 32nd international conference on machine learning*, volume 37 of *Proceedings of machine learning research*, pages 1613–1622, Lille, France, July 2015. PMLR. URL http://proceedings.mlr.press/v37/blundell15.html. tex.date-added: 2019-09-24 16:11:39 +0000 tex.date-modified: 2019-09-24 16:11:46 +0000 tex.pdf: http://proceedings.mlr.press/v37/blundell15.pdf.

References III

K. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Netw.*, 2(3):183–192, May 1989. ISSN 0893-6080. doi: 10.1016/0893-6080(89)90003-8. URL http://dx.doi.org/10.1016/0893-6080(89)90003-8. Number of pages: 10 Publisher: Elsevier Science Ltd. tex.acmid: 71290 tex.address: Oxford, UK, UK tex.date-added: 2019-09-19 05:15:10 +0000 tex.date-modified: 2019-09-19 05:15:17 +0000 tex.issue_date: 1989.

Yarin Gal. *Uncertainty in deep learning*. PhD thesis, PhD thesis, University of Cambridge, 2016. tex.date-added: 2019-09-19 11:45:00 +0000 tex.date-modified: 2019-09-24 16:02:09 +0000.

Yarin Gal and Zoubin Ghahramani. Bayesian convolutional neural networks with Bernoulli approximate variational inference. In *ICLR - workshop track*, 2016a.

References IV

Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1050–1059, New York, New York, USA, June 2016b. PMLR. URL http://proceedings.mlr.press/v48/gal16.html.

Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in neural information processing systems 29*, pages 1019–1027. Curran Associates, Inc., 2016c. URL http://papers.nips.cc/paper/6241-a-theoretically-grounded-application-of-dropout-in-rec pdf.

References V

Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P Vetrov, and Andrew G Wilson. Loss Surfaces, Mode Connectivity, and Fast Ensembling of DNNs. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 8789–8798. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/8095-loss-surfaces-mode-connectivity-and-fast-ensembling-of pdf.

Andrew Gelman, John Carlin, Hal Stern, David Dunson, Aki Vehtari, and Donald Rubin. *Bayesian Data Analysis*. Chapman and Hall/CRC, third edition, 2013.

References VI

Matthew D. Hoffman, David M. Blei, Chong Wang, and John Paisley. Stochastic variational inference. *J. Mach. Learn. Res.*, 14(1):1303–1347, May 2013. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2502581.2502622. Number of pages: 45 Publisher: JMLR.org tex.acmid: 2502622 tex.date-added: 2019-09-24 13:46:13 +0000 tex.date-modified: 2019-09-24 13:46:13 +0000 tex.issue_date: January 2013.

David J. C. MacKay. Bayesian interpolation. *Neural Comput.*, 4(3): 415–447, May 1992a. ISSN 0899-7667. doi: 10.1162/neco.1992.4.3.415. URL http://dx.doi.org/10.1162/neco.1992.4.3.415. Number of pages: 33 Publisher: MIT Press tex.acmid: 148163 tex.address: Cambridge, MA, USA tex.date-added: 2019-09-18 21:10:45 +0000 tex.date-modified: 2019-09-18 21:10:45 +0000 tex.issue_date: May 1992.

References VII

David JC MacKay. *Bayesian methods for adaptive models*. PhD thesis, California Institute of Technology, 1992b. tex.date-added: 2019-09-18 21:12:59 +0000 tex.date-modified: 2019-09-18 21:12:59 +0000.

David JC MacKay. Introduction to gaussian processes. *NATO ASI Series F Computer and Systems Sciences*, 168:133–166, 1998. Publisher: Springer Verlag tex.date-added: 2019-09-18 21:12:30 +0000 tex.date-modified: 2019-09-18 21:12:30 +0000.

Radford M. Neal. Priors for infinite networks. Technical report, Dept. of Computer Science, University of Toronto, 1994. tex.date-added: 2019-09-18 21:06:36 +0000 tex.date-modified: 2019-09-18 21:07:22 +0000.

Radford M. Neal. Priors for infinite networks. In *Bayesian learning for neural networks*, pages 29–53. Springer New York, New York, NY, 1996. ISBN 978-1-4612-0745-0.

References VIII

Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, D. Sculley, Sebastian Nowozin, Joshua Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model\textquotesingle s uncertainty? Evaluating predictive uncertainty under dataset shift. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 13991–14002. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/9547-can-you-trust-your-models-uncertainty-evaluating-predi pdf.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15 (56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

References IX

Michalis Titsias and Miguel Lázaro-Gredilla. Doubly Stochastic Variational Bayes for non-Conjugate Inference. In *International Conference on Machine Learning*, pages 1971–1979. PMLR, June 2014. URL http://proceedings.mlr.press/v32/titsias14.html. ISSN: 1938-7228.

Andrew Gordon Wilson. The case for Bayesian deep learning. *arXiv preprint arXiv:2001.10995*, 2020.

Andrew Gordon Wilson and Pavel Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. *arXiv preprint arXiv:2002.08791*, 2020.