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

Uncertainty

But I want to make a strong argument. So, let's choose perturbed parameters **very** carefully. Say we have a stochastic process that only generates instances of θ that are *well-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 $\Theta|\mathcal{D}$.

Uncertainty

But I want to make a strong argument. So, let's choose perturbed parameters **very** carefully. Say we have a stochastic process that only generates instances of θ that are *well-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 $\text{Var}(Q)$ 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.

Uncertainty

But I want to make a strong argument. So, let's choose perturbed parameters **very** carefully. Say we have a stochastic process that only generates instances of θ that are *well-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 $\text{Var}(Q)$ 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.

Intuitively, the NNs that power our model know little about the mapping from this particular x_* to probability distributions over output spaces. Maybe x_* is in a region of $\mathcal{X}$ that's barely represented in $\mathcal{D}$.

Uncertainty

But I want to make a strong argument. So, let's choose perturbed parameters **very** carefully. Say we have a stochastic process that only generates instances of θ that are *well-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 $\text{Var}(Q)$ 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.

Intuitively, the NNs that power our model know little about the mapping from this particular x_* to probability distributions over output spaces. 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 $\text{Var}(Y|x,\theta)$.

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

Uncertainty is quantified by $\text{Var}(Y_*|x_*,\mathcal{D})$.

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

- $\text{Var}(Y|X=x,\mathcal{D},\mathcal{M})$ is a measure of the uncertainty of a model $\mathcal{M}$ about Y given x taking into account all hypotheses supported by a set of observations $\mathcal{D}$;
- $\text{Var}(Y|X=x,\Theta=\theta,\mathcal{M})$ is the variance of the likelihood 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 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

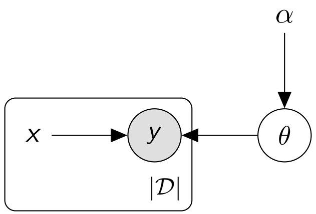
Bayes: what and why?

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!

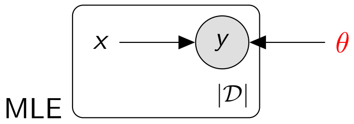
Observations are rvs, and *parameters* are rvs.

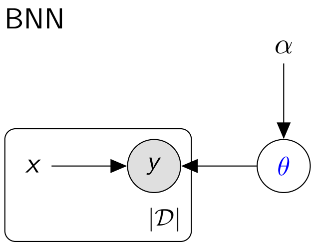
In a Bayesian model, data and parameters are no different. They are all given random treatment. The only substantial difference, being observed or not, has no effect in the theory. Read it this way: the set of principles is the same (axiomatic probability theory), there is no need for context-dependent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





- MLE: 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 *asymptotically*, under the assumption that you have infinite data and access to 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 though.

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 $\langle x, y \rangle$?

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. In ML we should be that kind of flexible. 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 in feature engineering you 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?

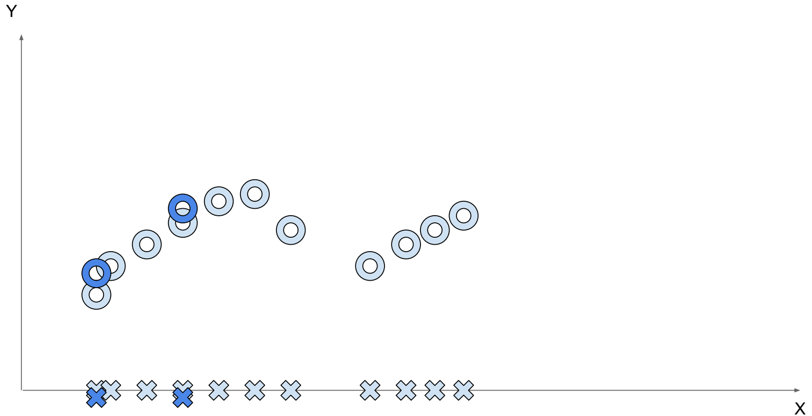
When a data point is well supported by θ, that is, $p(y_*|x_*, \theta)$ is high, we should ask ourselves, is θ even supported by the evidence we have?

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

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

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

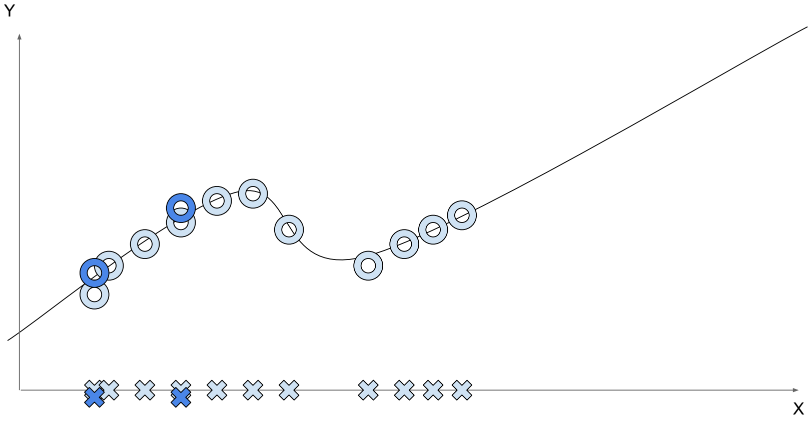
Uncertainty illustrated



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

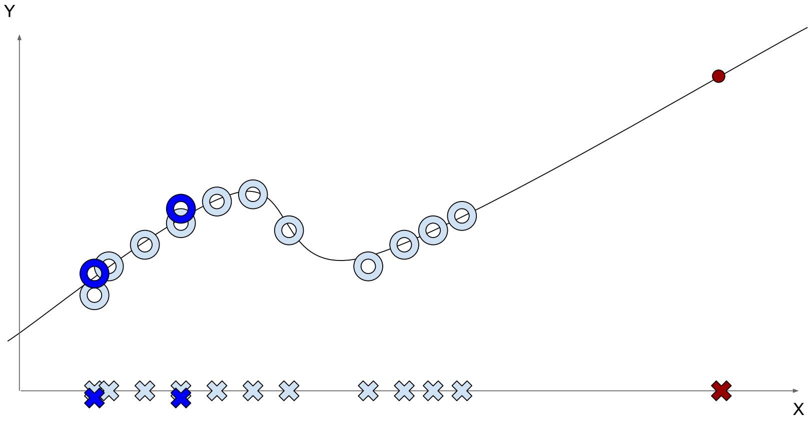


Let's approach with the help of NNs, i.e. $y = \mathrm{NN}(x;\theta)$

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

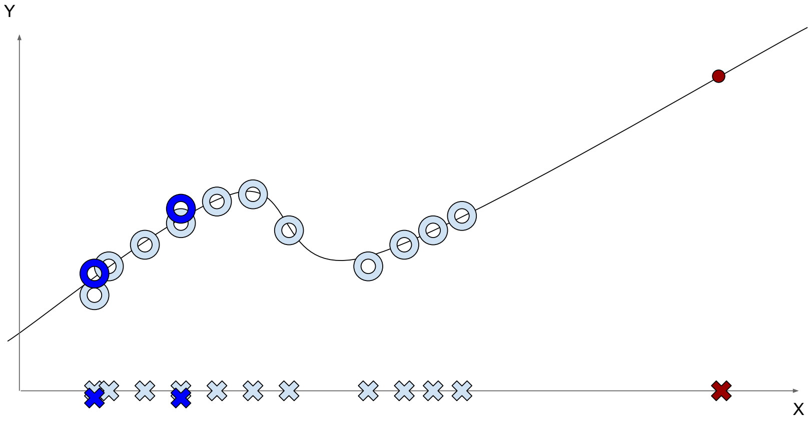


By design, it extrapolates predictions to unseen inputs, e.g. x_*

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

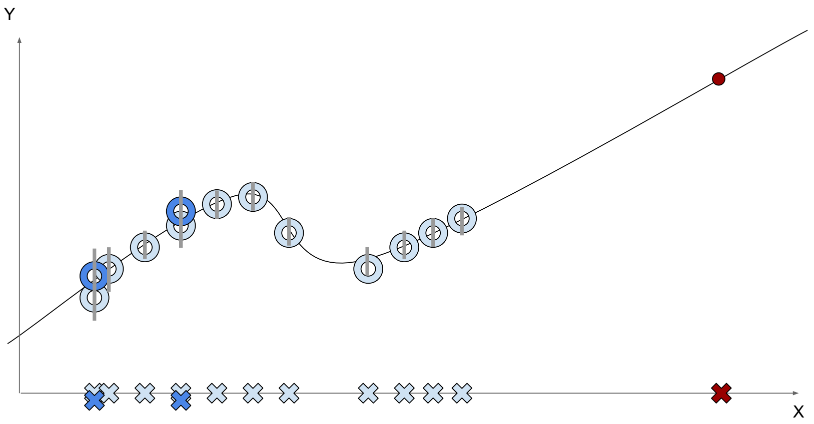


Can we trust our model given x_* is far from observations?

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

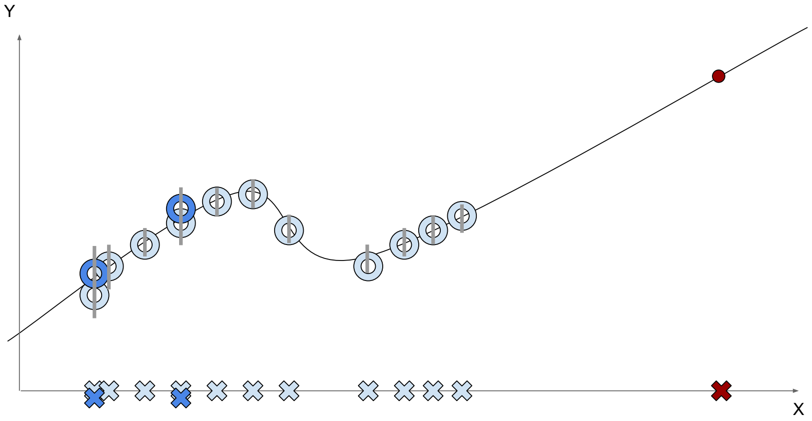


Let's fit Gaussians, i.e. $\mathcal{N}(\mu(x;\theta), \sigma(x;\theta)^2)$, around targets

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

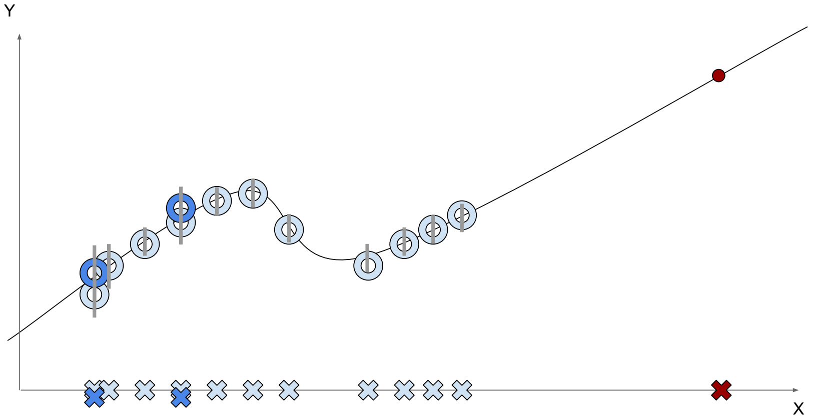


Note that we never observe much variability for a given input x

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

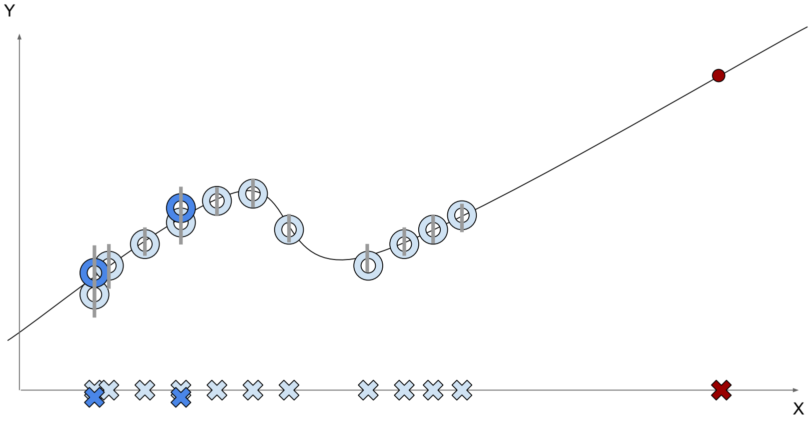


$\mu(x;\theta)$ learns to be on average close to every response for x

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

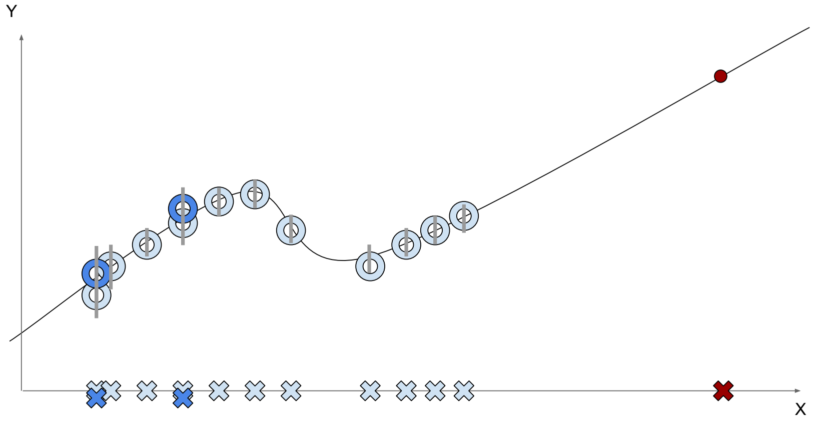


$\sigma(x;\theta)$ instead learns to cover all responses for x, but no more

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

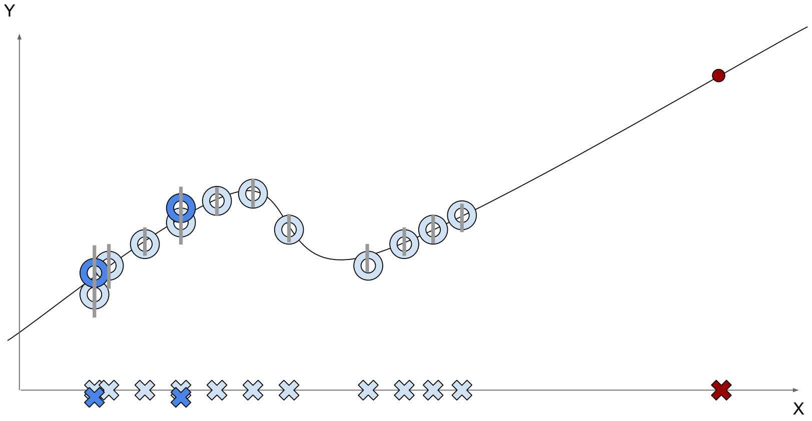


for MLE does not like covering more than observed responses

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

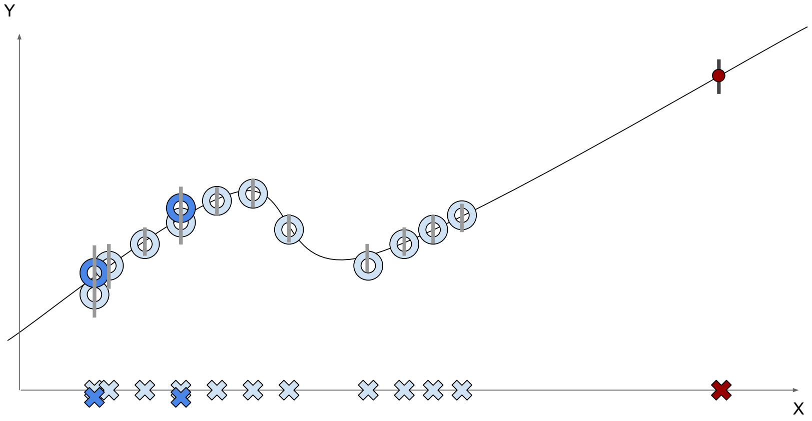


What is our expectation for $\sigma(x_*; \theta)$?

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

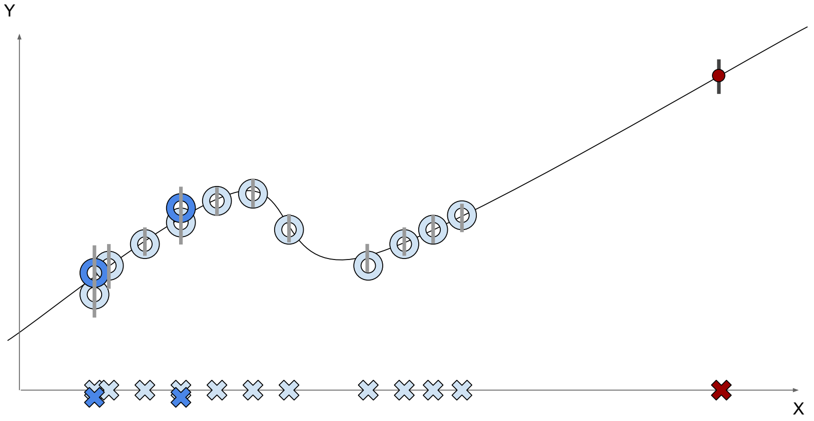


For all we know, $\sigma(\cdot;\theta)$ likes to predict small values

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

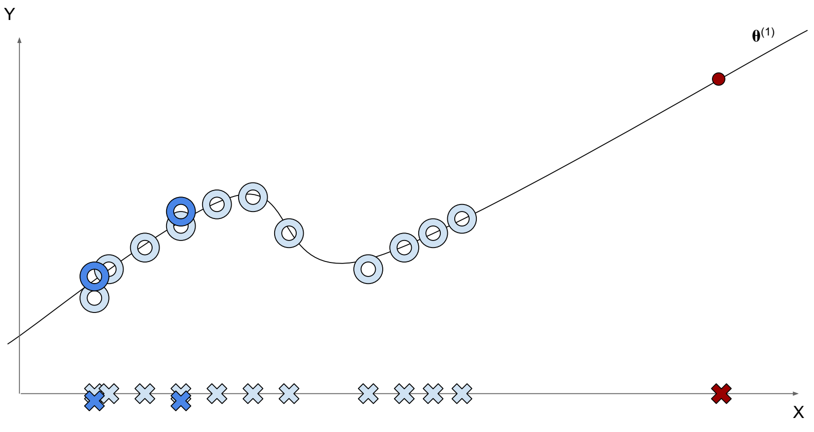


$\sigma(\cdot;\theta)$ seriously underestimates uncertainty for x_*

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

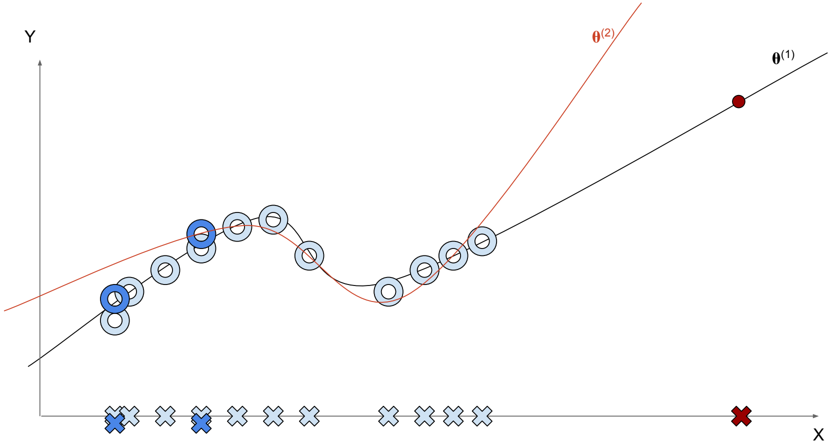


but what if, with probability $p(\theta^{(1)}|\mathcal{D})$, we consulted $\mu(x;\theta^{(1)})$?

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

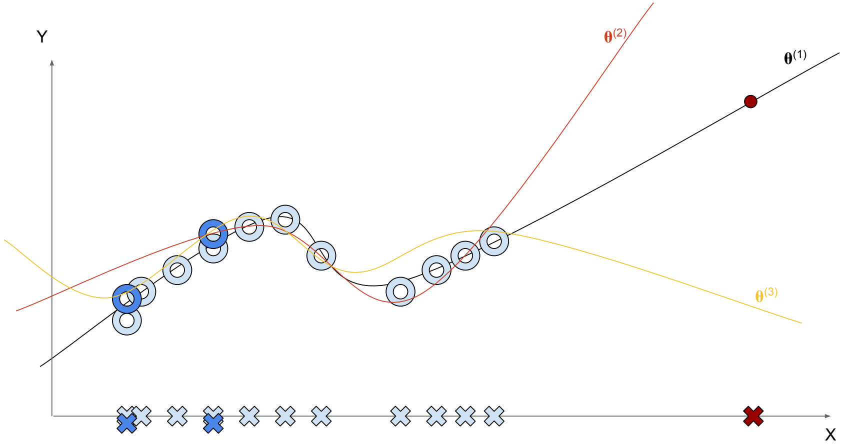


and $\mu(x; \theta^{(2)})$, with probability $p(\theta^{(2)}|\mathcal{D})$

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

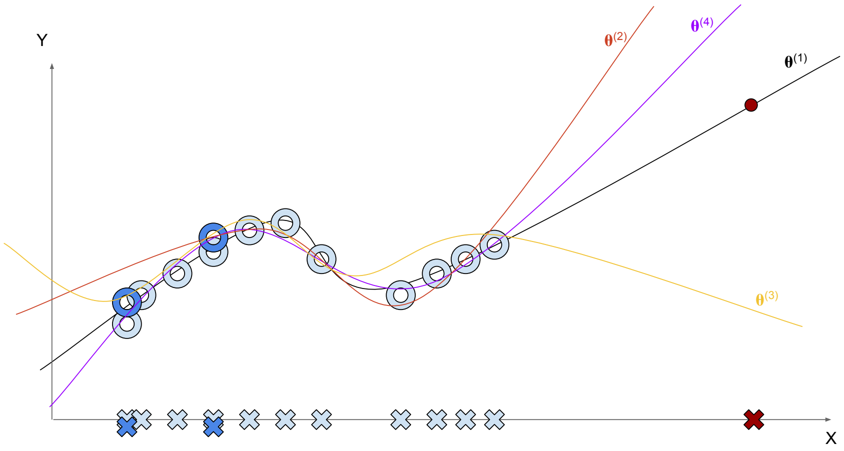


and $\mu(x; \theta^{(3)})$, with probability $p(\theta^{(3)}|\mathcal{D})$

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

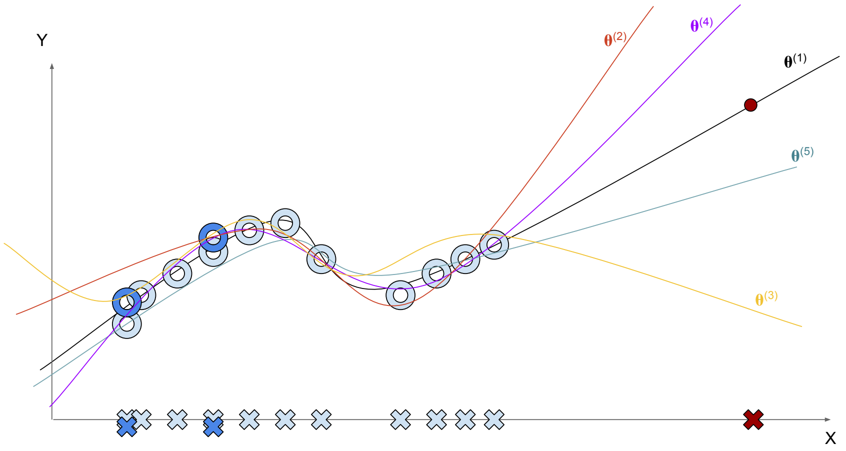


and $\mu(x; \theta^{(4)})$, with probability $p(\theta^{(4)}|\mathcal{D})$

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

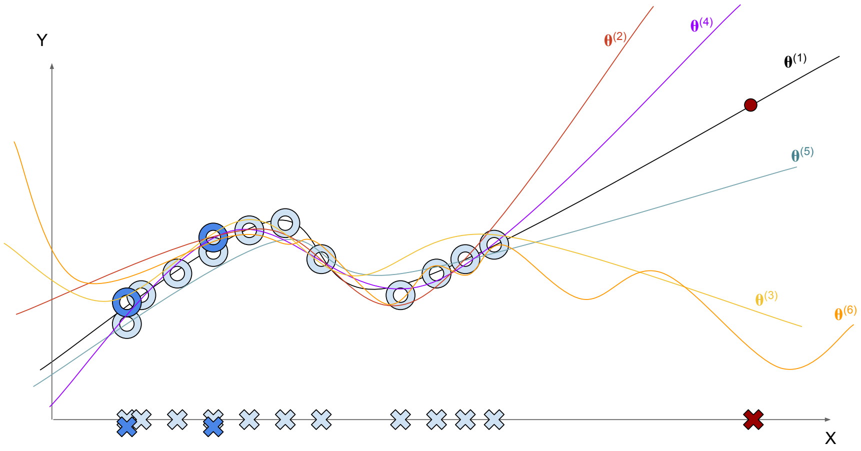


and $\mu(x; \theta^{(5)})$, with probability $p(\theta^{(5)}|\mathcal{D})$

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

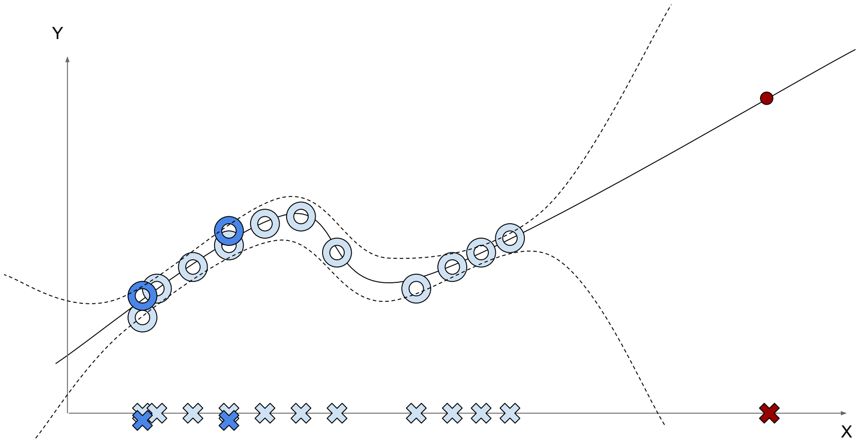


and $\mu(x; \theta^{(6)})$, with probability $p(\theta^{(6)}|\mathcal{D})$?

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).

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).

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):
$\hat{\mu}(x_*) = \frac{1}{K}\sum_{i=1}^{K} \mu(x_*; \theta^{(i)})$ for $\theta^{(i)} = \mathrm{MLE}(\mathcal{D})$

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).

But let's build intuition, and expose differences!

- Contrast this with stochasticity due to assessments of $\mu(x_*; \theta)$ 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):
$\hat{\mu}(x_*) = \frac{1}{K}\sum_{i=1}^{K} \mu(x_*; \theta^{(i)})$ for $\theta^{(i)} = \text{MLE}(\mathcal{D})$

Committing to $\hat{\mu}(x_*)$ disregards the actual spread of predictions hiding uncertainty rather than exposing it.

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).

But let's build intuition, and expose differences!

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

Bayesian Reasoning

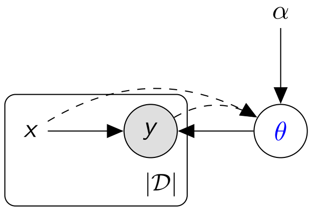
Reasoning with *parametric BNNs* involves averaging over parameters

- 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

We will discuss *nonparametric Bayesian models* later and will see that uncertainty estimates are based on yet more robust model assumptions.

Bayesian Inference

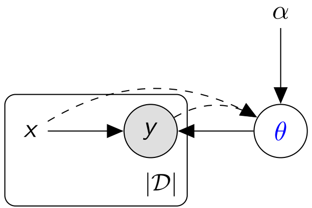
Posterior inference



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

Bayesian Inference

Posterior inference

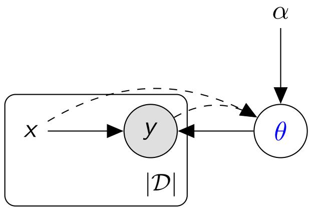


$$p(\theta|\mathcal{D}) = \frac{\overbrace{p(\theta)}^{\text{prior}}\,\overbrace{p(\mathcal{D}|\theta)}^{\text{likelihood}}}{\underbrace{p(\mathcal{D})}_{\text{evidence}}}$$

- The graphical model tells us that observations are independent of one another given θ. The dashed arrows 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



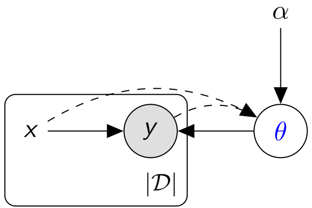
$$p(\theta|\mathcal{D}) = \frac{\overbrace{p(\theta)}^{\text{prior}}\ \overbrace{p(\mathcal{D}|\theta)}^{\text{likelihood}}}{\underbrace{p(\mathcal{D})}_{\text{evidence}}}$$

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

- The graphical model tells us that observations are independent of one another given θ. The dashed arrows 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



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

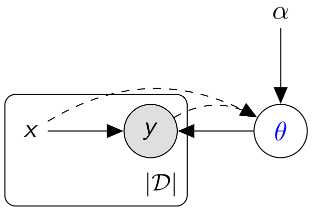
$$p(\mathcal{D}|\theta) = \prod_{\langle x,y\rangle\in\mathcal{D}} p(y|x,\theta)$$

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

- The graphical model tells us that observations are independent of one another given θ. The dashed arrows 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

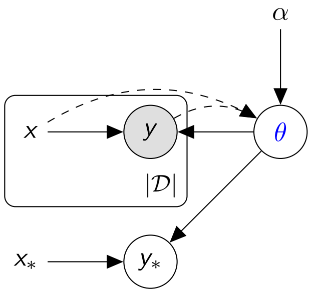


$$p(\theta|\mathcal{D}) = \frac{\overbrace{p(\theta)}^{\text{prior}}\ \overbrace{p(\mathcal{D}|\theta)}^{\text{likelihood}}}{\underbrace{p(\mathcal{D})}_{\text{evidence}}}$$

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

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

Posterior predictive distribution

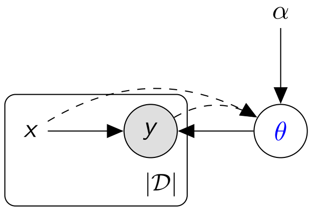


$$p(y_*|x_*,\mathcal{D}) =$$

- The graphical model tells us that observations are independent of one another given θ. The dashed arrows 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

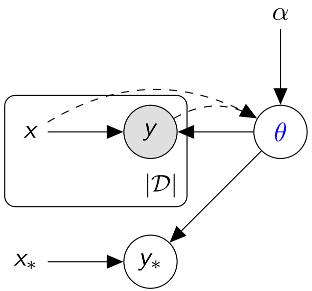


$$p(\theta|\mathcal{D}) = \frac{\overbrace{p(\theta)}^{\text{prior}}\,\overbrace{p(\mathcal{D}|\theta)}^{\text{likelihood}}}{\underbrace{p(\mathcal{D})}_{\text{evidence}}}$$

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

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

Posterior predictive distribution

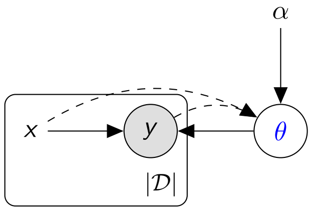


$$p(y_*|x_*,\mathcal{D}) = \int p(y_*,\theta|\mathcal{D},x_*)\mathrm{d}\theta$$

- The graphical model tells us that observations are independent of one another given θ. The dashed arrows 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

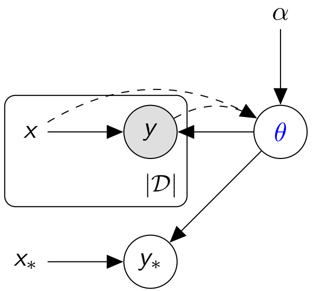


$$p(\theta|\mathcal{D}) = \underbrace{\frac{\overbrace{p(\theta)}^{\text{prior}}\ \overbrace{p(\mathcal{D}|\theta)}^{\text{likelihood}}}{p(\mathcal{D})}}_{\text{evidence}}$$

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

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

Posterior predictive distribution



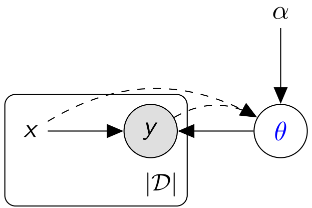
$$p(y_*|x_*,\mathcal{D}) = \int p(y_*,\theta|\mathcal{D},x_*)\mathrm{d}\theta$$
$$= \int p(\theta|\mathcal{D})p(y_*|x_*,\theta)\mathrm{d}\theta$$

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 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

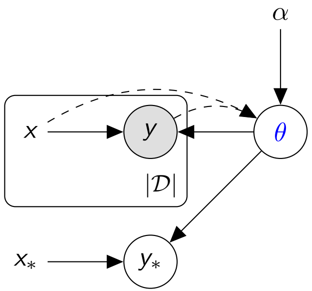


$$p(\theta|\mathcal{D}) = \frac{\overbrace{p(\theta)}^{\text{prior}}\,\overbrace{p(\mathcal{D}|\theta)}^{\text{likelihood}}}{\underbrace{p(\mathcal{D})}_{\text{evidence}}}$$

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

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

Posterior predictive distribution



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

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 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

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)

Outline

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.

MLE

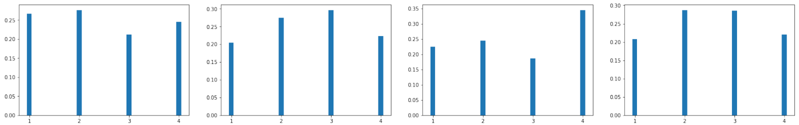
- parameters are **given**
- where do they come from?

Bayes

- parameters are rvs (there's no search for parameters, they come from the prior, stochastically)
- but it's worth asking, what does it mean to impose a Dirichlet prior on mixing coefficients π?
- what does it mean 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 $\pi \sim \text{Dir}(10 \times \mathbf{1}_K)$



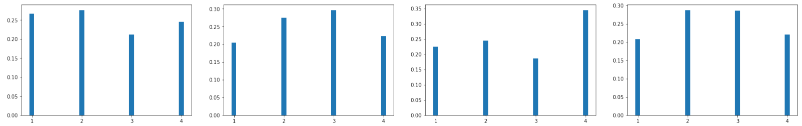
The question you have to think about is: Can we make any assumptions before observing data?

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

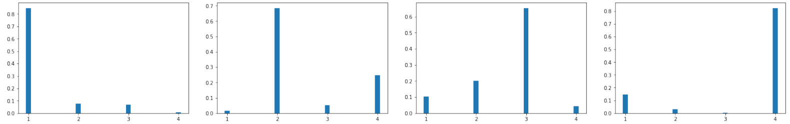
- Here you see 4 models that are likely under this prior. Each of these models gives every component of the mixture roughly the same opportunity to generate data points.

What makes good mixing coefficients?

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



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



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

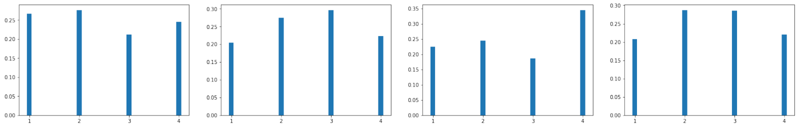
For example, let's see what happens as we vary our choice of prior for mixing coefficients

- 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.

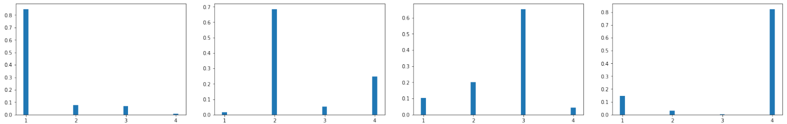
What makes good mixing coefficients?

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

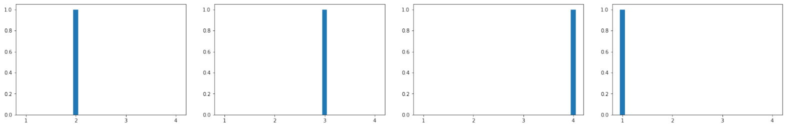
$\pi \sim \mathrm{Dir}(10 \times \mathbf{1}_K)$



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



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



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

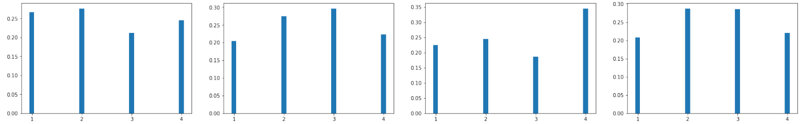
For example, let's see what happens as we vary our choice of prior for mixing coefficients

- 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).

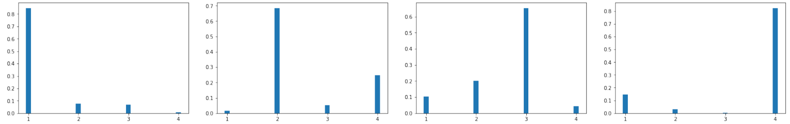
What makes good mixing coefficients?

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

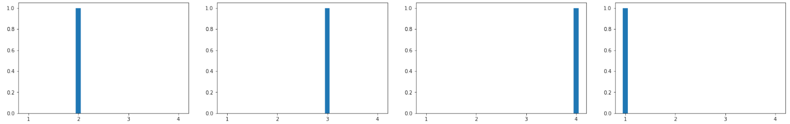
$\pi \sim \mathrm{Dir}(10 \times \mathbf{1}_K)$



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



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



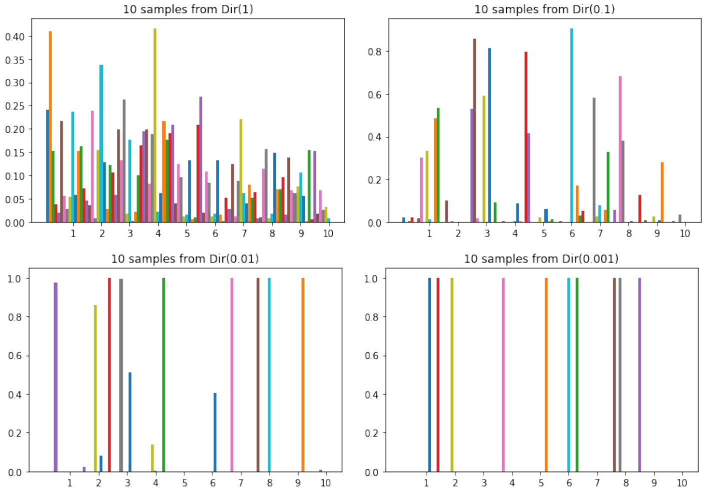
The question you have to think about is: Can we make any assumptions before observing data?

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 $\theta^{(k)}|\beta \sim \text{Dir}(\beta)$



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*?

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 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 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?*

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 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.

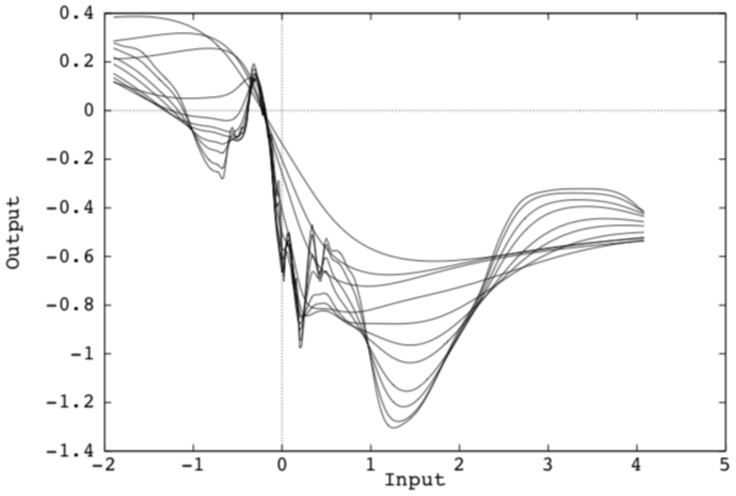
Random functions

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

Draws



Example from MacKay (1998)

Priors

Priors are not about making things random for no reason
they are about encoding assumptions, or inductive biases!

Let's turn to known priors over functions!

Gaussian Processes

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?

Gaussian Processes

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 $x^{(1)}, \dots, x^{(N)}$ the density $p(f(x^{(1)}), \dots, f(x^{(N)}))$ is a Gaussian.

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

GP prior

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. $x_i \stackrel{\text{def}}{=} x^{(i)}$.

Check the *excellent* Kernel Cookbook by David Duvenaud

GP prior

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. $x_i \stackrel{\text{def}}{=} x^{(i)}$. Similarly, $\mathbf{f} = \{f(x^{(1)}), \ldots, f(x^{(N)})\}$ is a collection of latent function assessments.

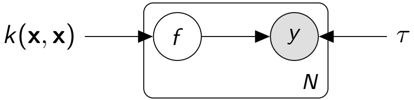
Check the *excellent* Kernel Cookbook by David Duvenaud

GP prior

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.

GP prior

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



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

GP Inference

Given a collection $\mathbf{y}, \mathbf{f} | \mathbf{x}$ of jointly Gaussian variables, what's the family of the marginal?

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

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

GP Inference

Given a collection $\mathbf{y}, \mathbf{f}|\mathbf{x}$ of jointly Gaussian variables, what's the family of the marginal?

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

Recall: marginals of a multivariate Gaussian are Gaussians!

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

GP Inference

Given a collection $\mathbf{y}, \mathbf{f}|\mathbf{x}$ of jointly Gaussian variables, what's the family of the marginal?

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

Recall: marginals of a multivariate Gaussian are Gaussians!

Thus what's the family of the posterior?

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

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

GP Inference

Given a collection $\mathbf{y}, \mathbf{f}|\mathbf{x}$ of jointly Gaussian variables, what's the family of the marginal?

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

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 a multivariate Gaussian yields a multivariate Guassian

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

Exact Inference with GPs

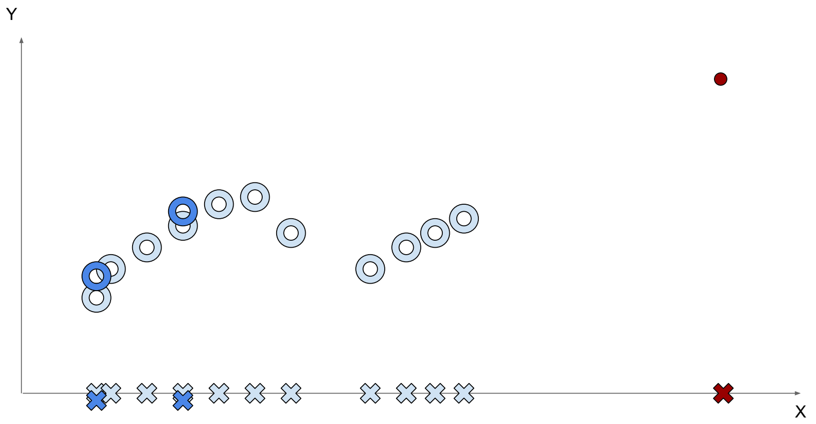
Posterior

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

Posterior predictive distribution:

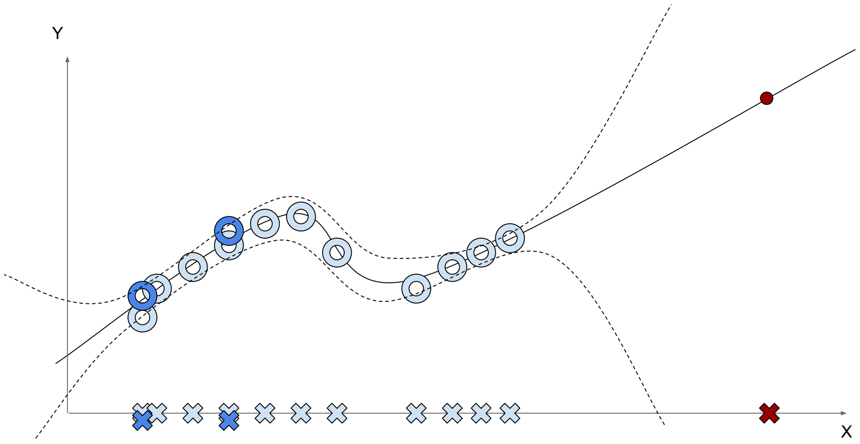
$$Y_*|x_*,\mathbf{x},\mathbf{y} \sim \mathcal{N}\big(k(x_*,\mathbf{x})(\mathbf{K} + \tau^{-1}\mathbf{I}_N)^{-1}\mathbf{y},\; k(x_*,x_*) + \tau^{-1} - k(x_*,\mathbf{x})(\mathbf{K} + \tau^{-1}\mathbf{I}_N)^{-1}k(x_*,\mathbf{x})^\top\big)$$

Uncertainty illustrated (revisited)



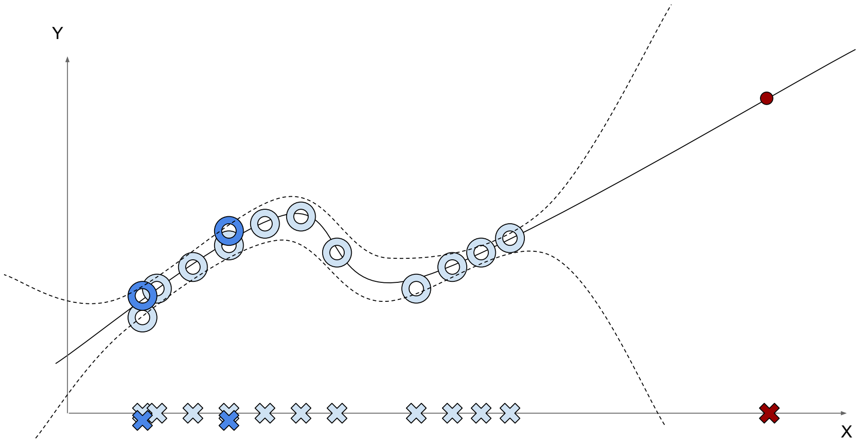
Let's get back to this

Uncertainty illustrated (revisited)



What if uncertainty depended on the distance to observations?

Uncertainty illustrated (revisited)



Kernels in GPs operationalise exactly 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: $F \sim \mathcal{G}P(0, k)$

GP inference: marginals and conditionals are Gaussians

Summary

Priors are as good as our understanding of what class of models they favour

NNs are meant to learn unknown functions

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

The effect of a parameter over the learned function is unclear

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 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

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

Why don't we always use GPs then?

Flexibility

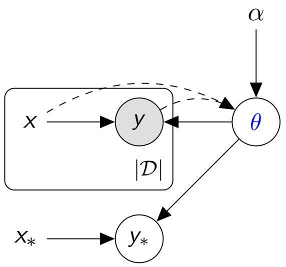
- kernels for text are fewer, less convenient, and less well-understood
- x can be very high-dimensional (and perhaps we have less intuitions to choose a kernel)

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

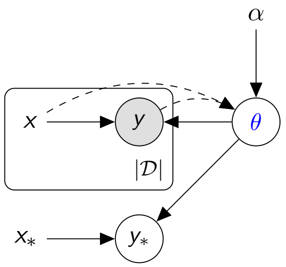
Let's then consider Bayesian inference for a BNN

This is essentially what we have to address



Let's then consider Bayesian inference for a BNN

This is essentially what we have to address

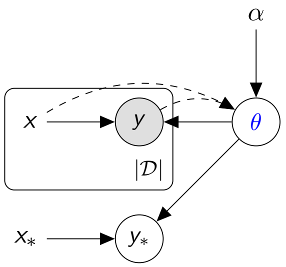


That is,

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

Let's then consider Bayesian inference for a BNN

This is essentially what we have to address



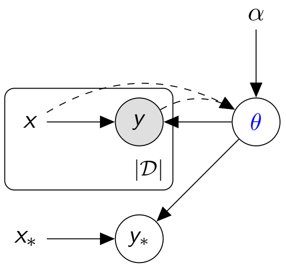
That is,

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

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

Let's then consider Bayesian inference for a BNN

This is essentially what we have to address



That is,

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

But $p(\theta|\mathcal{D}) = \int p(\theta|\alpha)p(\mathcal{D}|\theta)\mathrm{d}\theta$ is intractable
Let's learn a proxy $q(\theta|\lambda)$!

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 $p(\mathcal{D}, \theta)$.

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 $p(\mathcal{D}, \theta)$.

Evidence Lowerbound (ELBO)

Principle: choose an approximation that minimises KL-divergence

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

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

Evidence Lowerbound (ELBO)

Principle: choose an approximation that minimises KL-divergence

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

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

$$= \arg\min_{q(\theta)} \; \mathbb{E}_{q(\theta)}\left[\log \frac{q(\theta|\lambda)p(\mathcal{D})}{p(\theta,\mathcal{D})}\right] \qquad \text{definition of posterior}$$

Evidence Lowerbound (ELBO)

Principle: choose an approximation that minimises KL-divergence

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

Evidence Lowerbound (ELBO)

Principle: choose an approximation that minimises KL-divergence

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

Evidence Lowerbound (ELBO)

Principle: choose an approximation that minimises KL-divergence

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

Evidence Lowerbound (ELBO)

Principle: choose an approximation that minimises KL-divergence

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

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

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

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

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

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

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).

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).

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).

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).

Choosing λ

How should we choose λ?

Choosing λ

How should we choose λ?

How can we approach the following problem?

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

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?

Gradient-based optimisation for λ

We take steps in the direction that maximises the ELBO

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

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)!

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

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

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?

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

Stochastic gradients are allowed

Noisy, but unbiased, gradients:

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

Stochastic gradients are allowed

Noisy, but unbiased, gradients:

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

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

Stochastic gradients are allowed

Noisy, but unbiased, gradients:

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

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

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

Stochastic gradients are allowed

Noisy, but unbiased, gradients:

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

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

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

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

Stochastic gradients are allowed

Noisy, but unbiased, gradients:

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

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.

Challenge

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

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

Challenge

∇_λ ELBO =

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

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

Challenge

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

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

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

- 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?

Reparameterisation

Remember the *law of the unconscious statistician*?

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

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

Reparameterised Gradients

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

Reparameterised Gradients

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

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

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

Reparameterised Gradients

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

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

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

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

Reparameterised Gradients

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

∇_λ ELBO $=$

$$
\begin{aligned}
&N\mathbb{E}_l\left[\nabla_\lambda \mathbb{E}_{q(\theta|\lambda)}\left[\log p(y^{(l)}|\theta, x^{(l)})\right]\right] - \nabla_\lambda \mathrm{KL}(q(\theta|\lambda)||p(\theta)) \\
&= N\mathbb{E}_l\left[\nabla_\lambda \mathbb{E}_{\underline{\phi(\epsilon)}}\left[\log p(y^{(l)}|\underline{\theta = t(\epsilon, \lambda)}, x^{(l)})\right]\right] - \nabla_\lambda \mathrm{KL}(q(\theta|\lambda)||p(\theta)) \\
&= N\mathbb{E}_l\left[\mathbb{E}_{\phi(\epsilon)}\left[\nabla_\lambda \log p(y^{(l)}|t(\epsilon, \lambda), x^{(l)})\right]\right] - \nabla_\lambda \mathrm{KL}(q(\theta|\lambda)||p(\theta))
\end{aligned}
$$

Reparameterised Gradients

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

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

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

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

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

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

Reparameterised Gradient Estimate

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

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

Reparameterised Gradient Estimate

∇_λ ELBO $=$

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

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

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

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

Reparameterised Gradient Estimate

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

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

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

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

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

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

After training?

Training now gives you a point estimate for λ
so we are not training p, we are training q!

After training?

Training now gives you a point estimate for λ
so we are not training p, we are training q!

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?

Training now gives you a point estimate for λ
so we are not training p, we are training q!

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

After training?

Training now gives you a point estimate for λ
so we are not training p, we are training q!

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

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

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

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

Outline

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)

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

Many BNNs however can be seen as an approximation to a GP

- and the nonlinearities we employ correspond to a certain known kernel

Then let's see that variational inference for this model, using a pretty specific approximation $q(\theta|\lambda)$, turns dropout into approximate inference for an approximate GP.

The consequence is that we gain access to estimates of marginal likelihood and posterior predictive distribution

Notation

$\mathbf{w}$ is a C-dimensional column vector of parameters

$\mathbf{W} = [\mathbf{w}_r^\top]_{r=1}^R$ stacks row vectors into a $R \times C$ matrix

$\mathbf{x}$ is an I-dimensional input

$\mathbf{y}$ is an O-dimensional output

$\mathbf{x}_{1:N}, \mathbf{y}_{1:N}$ is a collection of input-output pairs

$\mathbf{y}_{:,d}$ gather the dth output of each observation

$\mathcal{N}(\mathbf{y}_{1:N}|\mathbf{m}_{1:N}, \mathbf{I}_N)$ denotes O independent multivariate Gaussians
i.e. $\prod_{d=1}^{O} \mathcal{N}(\mathbf{y}_{:,d}|\mathbf{m}_{:,d}, \mathbf{I}_N)$

A GP approximation

We specify a GP prior by specifying a kernel. Valid kernels can be composed into other valid kernels.

Formal properties of Kernels (Shawe-Taylor and Cristianini, 2004, Chapter 3)

A GP approximation

We specify a GP prior by specifying a kernel. Valid kernels can be composed into other valid kernels. For example:

$$k(\mathbf{x}, \mathbf{x}') = \int p(\mathbf{w})p(b)\sigma(\mathbf{w}^\top \mathbf{x} + b)\sigma(\mathbf{w}^\top \mathbf{x}' + b)\mathrm{d}\mathbf{w}\mathrm{d}b$$

This particular kernel is not tractable to assess: we cannot solve the integral for general $p(\mathbf{w}, b)$ and nonlinearity $\sigma(\cdot)$.

Formal properties of Kernels (Shawe-Taylor and Cristianini, 2004, Chapter 3)

A GP approximation

We specify a GP prior by specifying a kernel. Valid kernels can be composed into other valid kernels. For example:

$$k(\mathbf{x}, \mathbf{x}') = \int p(\mathbf{w})p(b)\sigma(\mathbf{w}^\top \mathbf{x} + b)\sigma(\mathbf{w}^\top \mathbf{x}' + b)\mathrm{d}\mathbf{w}\mathrm{d}b$$

This particular kernel is not tractable to assess: we cannot solve the integral for general $p(\mathbf{w}, b)$ and nonlinearity $\sigma(\cdot)$.

But we know MC estimation! For a fixed number of samples H

$$\hat{k}(\mathbf{x}, \mathbf{x}') = \frac{1}{H}\sum_{s=1}^{H} \sigma(\mathbf{w}_s^\top \mathbf{x} + b_s)\sigma(\mathbf{w}_s^\top \mathbf{x}' + b_s)$$

where $\mathbf{w}_s \sim p(\mathbf{w})$ and $b_s \sim p(b)$

Formal properties of Kernels (Shawe-Taylor and Cristianini, 2004, Chapter 3)

GP based on $\hat{k}$

We can immediately define a GP using this new kernel

$$\begin{aligned} b_s &\sim \mathcal{N}(0, l_0^{-2}) \\ \mathbf{b} &= [b_1, \ldots, b_H]^\top \\ \mathbf{w}_s &\sim \mathcal{N}(0, l^{-2}\mathbf{I}_I) \\ \mathbf{W}_1 &= [\mathbf{w}_s^\top]_{s=1}^H \end{aligned}$$

$$\begin{aligned} \hat{\mathbf{K}} &= \hat{k}(\mathbf{x}_{1:N}, \mathbf{x}_{1:N}) \\ \mathbf{F}|\mathbf{x}, \mathbf{W}_1, \mathbf{b} &\sim \mathcal{N}(\mathbf{0}, \hat{\mathbf{K}}) \\ \mathbf{Y}|\mathbf{f} &\sim \mathcal{N}(\mathbf{f}, \tau^{-1}\mathbf{I}_N) \end{aligned}$$

Recall that $\hat{K}_{ij} = \frac{1}{H}\sum_{s=1}^{H} \sigma(\mathbf{w}_s^\top \mathbf{x}_i + b)\sigma(\mathbf{w}_s^\top \mathbf{x}_j + b)$

GP based on $\hat{k}$

We can immediately define a GP using this new kernel

$$
\begin{aligned}
b_s &\sim \mathcal{N}(0, l_0^{-2}) \\
\mathbf{b} &= [b_1, \ldots, b_H]^\top \\
\mathbf{w}_s &\sim \mathcal{N}(0, l^{-2}\mathbf{I}_I) \\
\mathbf{W}_1 &= [\mathbf{w}_s^\top]_{s=1}^H
\end{aligned}
\qquad
\begin{aligned}
\hat{\mathbf{K}} &= \hat{k}(\mathbf{x}_{1:N}, \mathbf{x}_{1:N}) \\
\mathbf{F}|\mathbf{x}, \mathbf{W}_1, \mathbf{b} &\sim \mathcal{N}(\mathbf{0}, \hat{\mathbf{K}}) \\
\mathbf{Y}|\mathbf{f} &\sim \mathcal{N}(\mathbf{f}, \tau^{-1}\mathbf{I}_N)
\end{aligned}
$$

Note that we have essentially parameterised our kernel and imposed a prior on the kernel parameters.

For any given parameter configuration $\mathbf{W}_1$ and $\mathbf{b}$, we get a GP:

$$
p(\mathbf{y}_{1:N}|\mathbf{x}_{1:N}) = \int p(\mathbf{b})p(\mathbf{W}_1)p(\mathbf{y}_{1:N}|\mathbf{f}_{1:N})p(\mathbf{f}_{1:N}|\mathbf{W}_1, \mathbf{b}, \mathbf{x}_{1:N})\mathrm{d}\mathbf{f}_{1:N}\mathrm{d}\mathbf{W}_1\mathrm{d}\mathbf{b}
$$

Recall that $\hat{K}_{ij} = \frac{1}{H}\sum_{s=1}^H \sigma(\mathbf{w}_s^\top \mathbf{x}_i + b)\sigma(\mathbf{w}_s^\top \mathbf{x}_j + b)$

Regression vs Classification

Regression

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

Classification

$$\mathbf{Y}|\mathbf{f} \sim \mathcal{N}(\mathbf{f}, 0\mathbf{I}_N)$$
$$C|\mathbf{y} \sim \text{Cat}(\text{softmax}(\mathbf{y}))$$

- the GP is essentially inducing a distribution over logits
- note the change of notation (to be closer to Gal's), here **y** is not an observation, c is

Marginalise $\mathbf{f}$

We can marginalise $\mathbf{f}$ for an assignment of $\mathbf{W}_1$ and $\mathbf{b}$

Can you show that $\hat{\mathbf{K}} = \Phi\Phi^\top$?

Marginalise $\mathbf{f}$

We can marginalise $\mathbf{f}$ for an assignment of $\mathbf{W}_1$ and $\mathbf{b}$

Note that $\hat{\mathbf{K}} = \Phi\Phi^\top$ for $\Phi = [\phi(\mathbf{x}_n, \mathbf{W}_1, \mathbf{b})^\top]_{n=1}^N$
with feature vectors $\phi(\mathbf{x}, \mathbf{W}_1, \mathbf{b}) = \sqrt{1/H}\sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b})$

Can you show that $\hat{\mathbf{K}} = \Phi\Phi^\top$?

Marginalise $\mathbf{f}$

We can marginalise $\mathbf{f}$ for an assignment of $\mathbf{W}_1$ and $\mathbf{b}$

Note that $\hat{\mathbf{K}} = \Phi\Phi^\top$ for $\Phi = [\phi(\mathbf{x}_n, \mathbf{W}_1, \mathbf{b})^\top]_{n=1}^N$
with feature vectors $\phi(\mathbf{x}, \mathbf{W}_1, \mathbf{b}) = \sqrt{1/H}\sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b})$

From marginalisation of a subset of jointly Gaussian variables

$$\begin{aligned} p(\mathbf{y}_{1:N}|\mathbf{x}_{1:N}) &= \int p(\mathbf{W}_1)p(\mathbf{b})p(\mathbf{y}_{1:N}|\mathbf{f}_{1:N})p(\mathbf{f}_{1:N}|\mathbf{W}_1, \mathbf{b}, \mathbf{x}_{1:N})\mathrm{d}\mathbf{f}_{1:N}\mathrm{d}\mathbf{W}_1\mathrm{d}\mathbf{b} \\ &= \int \mathcal{N}(\mathbf{y}_{1:N}|\mathbf{0}, \Phi\Phi^\top + \tau^{-1}\mathbf{I}_N)p(\mathbf{W}_1)p(\mathbf{b})\mathrm{d}\mathbf{W}_1\mathrm{d}\mathbf{b} \end{aligned}$$

We have "parameters", but note the model remains non-parametric
complexity (capacity) adjusts with data size

Can you show that $\hat{\mathbf{K}} = \Phi\Phi^\top$?

Parametric assumption

Recall that every Gaussian is the marginal of some other Gaussian.

Parametric assumption

Recall that every Gaussian is the marginal of some other Gaussian.

In particular, it's true that

$$\mathcal{N}(\mathbf{y}_{:,d}|\mathbf{0}, \Phi\Phi^\top + \tau^{-1}\mathbf{I}_N) = \int \mathcal{N}(\mathbf{y}_{:,d}|\Phi\mathbf{w}_d, \tau^{-1}\mathbf{I}_N)\mathcal{N}(\mathbf{w}_d|\mathbf{0}, \mathbf{I}_H)\mathrm{d}\mathbf{w}_d$$

where $\mathbf{w}_d \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_H)$ is an H-dimensional auxiliary random vector

Parametric assumption

Recall that every Gaussian is the marginal of some other Gaussian.

In particular, it's true that

$$\mathcal{N}(\mathbf{y}_{:,d}|\mathbf{0}, \Phi\Phi^\top + \tau^{-1}\mathbf{I}_N) = \int \mathcal{N}(\mathbf{y}_{:,d}|\Phi\mathbf{w}_d, \tau^{-1}\mathbf{I}_N)\mathcal{N}(\mathbf{w}_d|\mathbf{0}, \mathbf{I}_H)\mathrm{d}\mathbf{w}_d$$

where $\mathbf{w}_d \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_H)$ is an H-dimensional auxiliary random vector

We introduce O such vectors, $\mathbf{W}_2 = [\mathbf{w}_d^\top]_{d=1}^O$ each $\mathbf{w}_d \sim p(\mathbf{w})$

$$\begin{aligned} p(\mathbf{y}_{1:N}|\mathbf{x}_{1:N}) &= \int \mathcal{N}(\mathbf{y}_{1:N}|\mathbf{0}, \Phi\Phi^\top + \tau^{-1}\mathbf{I}_N)p(\mathbf{W}_1)p(\mathbf{b})\mathrm{d}\mathbf{W}_1\mathrm{d}\mathbf{b} \\ &= \int p(\mathbf{y}_{1:N}|\mathbf{x}_{1:N}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b})p(\mathbf{W}_1)p(\mathbf{b})p(\mathbf{W}_2)\mathrm{d}\mathbf{W}_2\mathrm{d}\mathbf{W}_1\mathrm{d}\mathbf{b} \end{aligned}$$

Digest

In the “first layer” we project inputs to H-dimensional *feature vectors* and apply a nonlinearity $\sigma(\cdot)$. This is a “hidden layer”.

The parameters $\mathbf{W}_1$ and $\mathbf{b}$ of this projection are stochastic.

Digest

In the "first layer" we project inputs to H-dimensional *feature vectors* and apply a nonlinearity $\sigma(\cdot)$. This is a "hidden layer".

The parameters $\mathbf{W}_1$ and $\mathbf{b}$ of this projection are stochastic.

In the "second layer" we decompose the covariance matrix using an H-dimensional vector (regardless of data size N). We do so independently for each of the O outputs of the model. This is an "output layer".

Again the parameters $\mathbf{W}_2$ of this decomposition are stochastic.

Digest

In the "first layer" we project inputs to H-dimensional *feature vectors* and apply a nonlinearity $\sigma(\cdot)$. This is a "hidden layer".

The parameters $\mathbf{W}_1$ and $\mathbf{b}$ of this projection are stochastic.

In the "second layer" we decompose the covariance matrix using an H-dimensional vector (regardless of data size N). We do so independently for each of the O outputs of the model. This is an "output layer".

Again the parameters $\mathbf{W}_2$ of this decomposition are stochastic.

In the limit of an infinite hidden layer, marginalising over the stochastic parameters gives us a GP likelihood.

The induced kernel depends on the non-linearity $\sigma(\cdot)$

A parametric approximation to a GP

A BNN with Gaussian priors over parameters is a parametric approximation to a GP

$$p(\mathbf{y}_{1:N}|\mathbf{x}_{1:N}) = \int p(\mathbf{y}_{1:N}|\mathbf{x}_{1:N}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b})p(\mathbf{W}_1, \mathbf{W}_2, \mathbf{b})\mathrm{d}\mathbf{W}_2\mathrm{d}\mathbf{W}_1\mathrm{d}\mathbf{b}$$

Obviously the marginal is intractable, after all this is a BNN!

This is true for tanh, relu, step functions, sigmoid, for example.

VI

Recipe

- propose a parametric proxy $q(\theta|\lambda)$
  $\theta = \{\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}\}$
- which you can reparameterise
- choose λ to maximise the ELBO via stochastic gradient-based optimisation
- now because we count on autodiff, make sure you use differentiable nonlinearities (step function is no longer an option)

Mean field

Independence across parameter groups

$$q(\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}|\lambda) = q(\mathbf{W}_1|\lambda_1)q(\mathbf{W}_2|\lambda_2)q(\mathbf{b}|\lambda_b)$$

Mean field

Independence across parameter groups

$$q(\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}|\lambda) = q(\mathbf{W}_1|\lambda_1)q(\mathbf{W}_2|\lambda_2)q(\mathbf{b}|\lambda_b)$$

Diagonal Gaussian for biases

$$q(\mathbf{b}|\lambda) = \mathcal{N}(\mathbf{b}|\mathbf{m}, \mathrm{diag}(\boldsymbol{\sigma}^2))$$ nicely reparameterisable!

Mean field

Independence across parameter groups

$$q(\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}|\lambda) = q(\mathbf{W}_1|\lambda_1)q(\mathbf{W}_2|\lambda_2)q(\mathbf{b}|\lambda_b)$$

Diagonal Gaussian for biases

$$q(\mathbf{b}|\lambda) = \mathcal{N}(\mathbf{b}|\mathbf{m}, \mathrm{diag}(\boldsymbol{\sigma}^2))$$ nicely reparameterisable!

Mixture of Gaussians posterior with $p \in (0, 1)$

$$q(\mathbf{w}_r|\lambda) = p\mathcal{N}(\mathbf{w}_r|\mathbf{m}_r, \mathrm{diag}(\boldsymbol{\sigma}^2)) + (1-p)\mathcal{N}(\mathbf{w}_r|\mathbf{0}, \mathrm{diag}(\boldsymbol{\sigma}^2))$$

Mean field

Independence across parameter groups

$$q(\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}|\lambda) = q(\mathbf{W}_1|\lambda_1)q(\mathbf{W}_2|\lambda_2)q(\mathbf{b}|\lambda_b)$$

Diagonal Gaussian for biases

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

Mixture of Gaussians posterior with $p \in (0, 1)$

$$q(\mathbf{w}_r|\lambda) = p\mathcal{N}(\mathbf{w}_r|\mathbf{m}_r, \mathrm{diag}(\boldsymbol{\sigma}^2)) + (1-p)\mathcal{N}(\mathbf{w}_r|\mathbf{0}, \mathrm{diag}(\boldsymbol{\sigma}^2))$$

Can we reparameterise samples from this posterior?

Mean field

Independence across parameter groups

$$q(\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}|\lambda) = q(\mathbf{W}_1|\lambda_1)q(\mathbf{W}_2|\lambda_2)q(\mathbf{b}|\lambda_b)$$

Diagonal Gaussian for biases

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

Mixture of Gaussians posterior with $p \in (0, 1)$

$$q(\mathbf{w}_r|\lambda) = p\mathcal{N}(\mathbf{w}_r|\mathbf{m}_r, \mathrm{diag}(\boldsymbol{\sigma}^2)) + (1-p)\mathcal{N}(\mathbf{w}_r|\mathbf{0}, \mathrm{diag}(\boldsymbol{\sigma}^2))$$

Can we reparameterise samples from this posterior?

- $\mathbf{w}_r = z(\mathbf{m}_r + \boldsymbol{\sigma} \odot \epsilon) + (1-z)\boldsymbol{\sigma} \odot \epsilon$
  for $Z \sim \mathrm{Bern}(p)$ and $\epsilon \sim \phi(\epsilon)$

Mixture of Deltas

Let $\boldsymbol{\sigma} \to \mathbf{0}$

- from mixture of Gaussians to mixture of Deltas

$$
\begin{aligned}
Z &\sim \text{Bern}(p) \\
\mathbf{w}_r &= \begin{cases} \mathbf{m}_r + \mathbf{0} \odot \epsilon & \text{if } z = 1 \\ (1-z)\mathbf{0} \odot \epsilon & \text{if } z = 0 \end{cases} \\
&= z\mathbf{m}_r
\end{aligned}
$$

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

Mixture of Deltas

Let $\boldsymbol{\sigma} \to \mathbf{0}$

- from mixture of Gaussians to mixture of Deltas

$$Z \sim \mathrm{Bern}(p)$$

$$\mathbf{w}_r = \begin{cases} \mathbf{m}_r + \mathbf{0} \odot \epsilon & \text{if } z = 1 \\ (1 - z)\mathbf{0} \odot \epsilon & \text{if } z = 0 \end{cases}$$

$$= z\mathbf{m}_r$$

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

The affine transform in fully connected layers becomes

$$([(z_s \mathbf{m}_s)^\top]_{s=1}^H)\mathbf{x} + \mathbf{m} = ([\mathbf{m}_s^\top]_{s=1}^H)(\mathbf{z} \odot x) + \mathbf{m}$$

with probability p, we essentially drop inputs

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

ELBO

Recall the ELBO

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

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 $Z_{2,s}\sim\mathrm{Bern}(p)$

ELBO

Recall the ELBO

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

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

ELBO

Recall the ELBO

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

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

ELBO

Recall the ELBO

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

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 can be nicely approximated by L_2 on λ

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 (we assume a degenerate Gaussian for $Y|f$)

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

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, generally we don't because Bernoulli sampling is nondifferentiable and thus we have to tune the parameter by hand.

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

- 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

Extensions

Note a few things

- 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

Extensions

Note a few things

- 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

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

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 (more correlations)
- 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)

References I

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.

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.

References II

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.

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:

References III

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.

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.

References IV

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.

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

References V

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 VI

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 VII

John Shawe-Taylor and Nello Cristianini. *Kernel methods for pattern analysis*. Cambridge University Press, New York, NY, USA, 2004. ISBN 0-521-81397-2. tex.date-added: 2019-09-25 04:24:57 +0000 tex.date-modified: 2019-09-25 04:25:07 +0000.

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.

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.