

Uncertainty in Machine Learning

Ehsan Abbasnejad



What is uncertainty in machine learning

We make observations using the sensors in the world (e.g. camera)

Based on the observations, we intend to make decisions

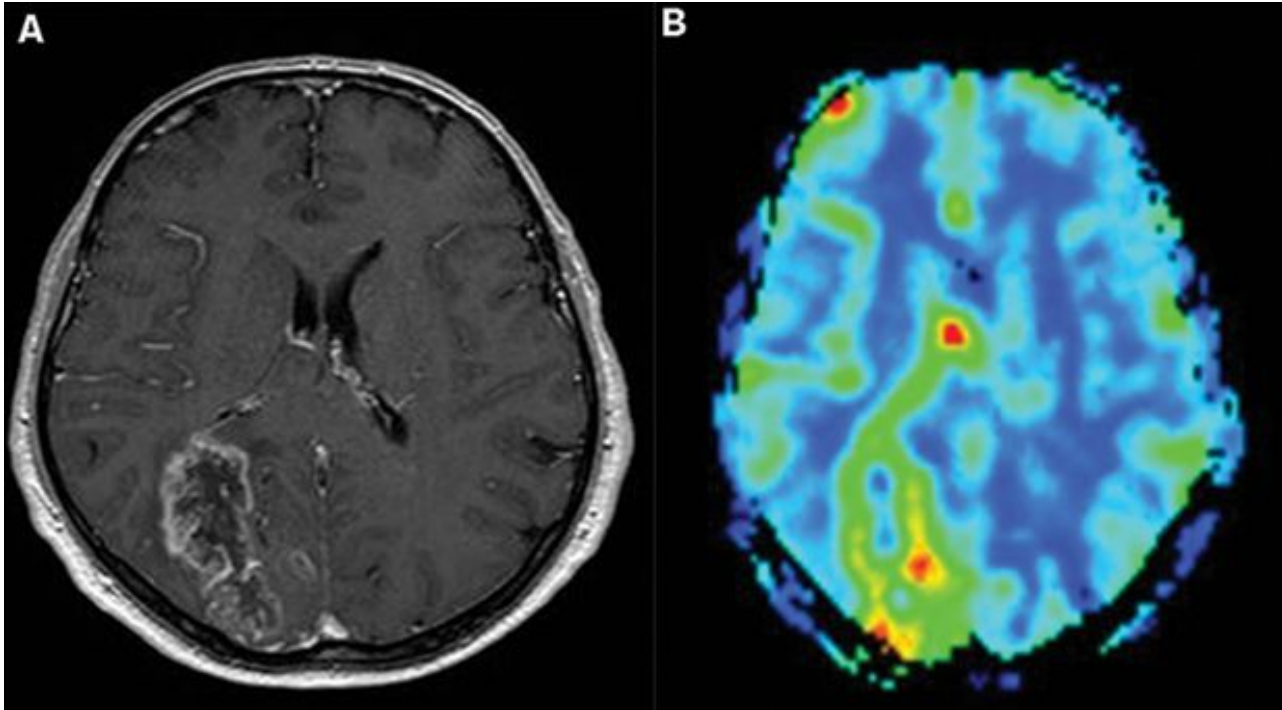
Given the same observations, the decision should be the same

However,

- The world changes, observations change, our sensors change, the output should not change!
- We'd like to know how confident we can be about the decisions

Why Uncertainty is important?

Medical diagnostics



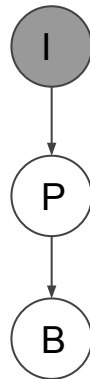
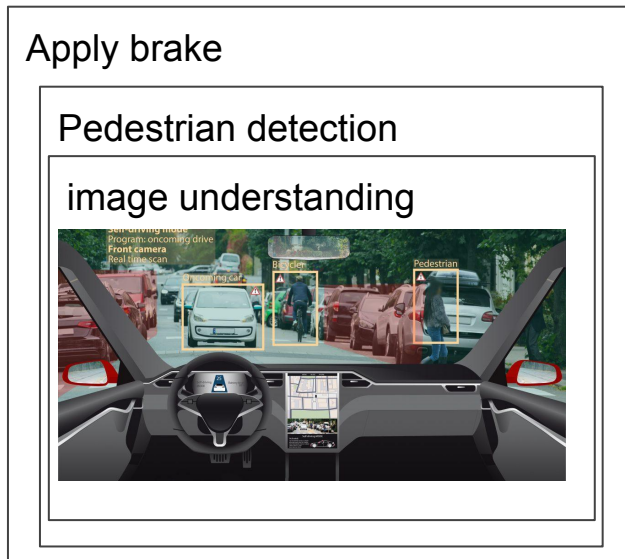
Why Uncertainty is important?

Imagine you are designing the vision system for an autonomous vehicle



Why Uncertainty is important?

Applications that require reasoning in earlier stages



What is uncertainty in machine learning

We build models for predictions, can we **trust** them? Are they **certain**?

What is uncertainty in machine learning

Many applications of machine learning depend on good estimation of the uncertainty:

- Forecasting
- Decision making
- Learning from limited, noisy, and missing data
- Learning complex personalised models
- Data compression
- Automating scientific modelling, discovery, and experiment design

Where uncertainty comes from?

Remember the machine learning's objective: minimize the **expected loss**

$$\begin{aligned}\min_{\theta} \mathbb{E}_{\mathbf{x}, y}[\ell(h(\mathbf{x}; \theta), y)] &= \int \ell(h(\mathbf{x}; \theta), y) dp^*(\mathbf{x}, y) \\ &\approx \frac{1}{n} \sum_{i=1}^n \ell(h(\mathbf{x}_i; \theta), y_i) \quad (\mathbf{x}_i, y_i) \sim p^*(\mathbf{x}, y)\end{aligned}$$

Uncertainty in data
(Aleatoric)

Uncertainty in the model
(Epistemic)

When the hypothesis function class is “simple” we can build generalization bound that underscore our confidence in average prediction

Where uncertainty comes from?

Alternatively, the probabilistic view:

$$\begin{aligned} \min_{\theta} \mathbb{E}_{\mathbf{x}, y} [-\log(p(y|\mathbf{x}, \theta))] \\ \approx \frac{1}{n} \sum_{i=1}^n -\log(p(y_i|\mathbf{x}_i, \theta)) \quad (\mathbf{x}_i, y_i) \sim p^*(\mathbf{x}, y) \end{aligned}$$

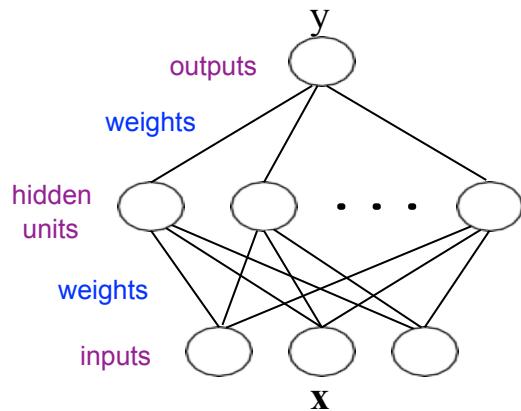
Uncertainty in data
(Aleatoric)

Uncertainty in the model
(Epistemic)

Which is,

$$\max_{\theta} \frac{1}{n} \sum_{i=1}^n \log(p(y_i|\mathbf{x}_i, \theta)) \quad (\mathbf{x}_i, y_i) \sim p^*(\mathbf{x}, y)$$

WHAT IS A NEURAL NETWORK?



Neural network is a parameterized function

Use a neural network to model the probability

Parameters θ are weights of neural net.

Feedforward neural nets model $p(y|\mathbf{x}, \theta)$ as a nonlinear function of θ and \mathbf{x} .

Samples from the true $(\mathbf{x}_i, y_i) \sim p^*(\mathbf{x}, y)$

distribution are the observations: $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$

Multilayer / deep neural networks model the overall function as a composition of functions (layers).

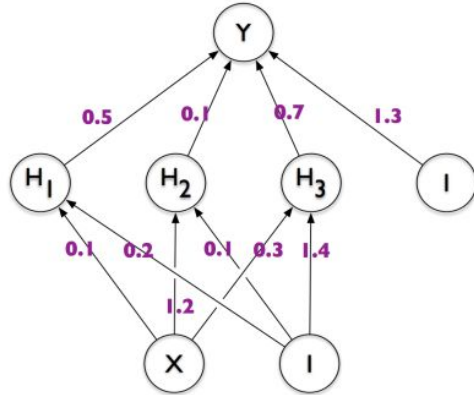
Usually trained to maximise likelihood (or penalised likelihood).

Point Estimate of Neural Nets: Maximum likelihood Estimate (MLE)

- The weights are obtained by minimizing the expected loss
- It assumes the state of the world is realized by a “single” parameter
- We assume the samples of observations are independent

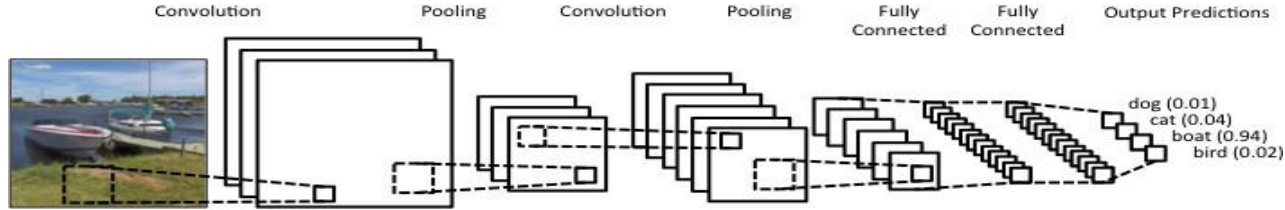
$$\max_{\theta} \mathbb{E}[\log(p(\mathcal{D}|\theta))]$$

$$\max_{\theta} \frac{1}{n} \sum_{i=1}^n \log(p(y_i | \mathbf{x}_i, \theta))$$



I'll use “minimizing loss” and “maximizing the likelihood” interchangeably.

DEEP LEARNING



Deep learning systems are neural network models similar to those popular in the '80s and '90s, with:

- Architectural and algorithmic **innovations** (e.g. many layers, ReLUs, better initialisation and learning rates, dropout, LSTMs, ...)
- Vastly larger **data** sets (web-scale)
- Vastly larger-scale **compute** resources (GPUs, cloud)
- Much better **software** tools (PyTorch, TensorFlow, MxNet, etc)
- Vastly increased industry **investment** and **media hype**

LIMITATIONS OF DEEP LEARNING

Neural networks and deep learning systems give amazing performance on many benchmark tasks, but they are generally:

- Very **data hungry** (e.g. often millions of examples)
- Very **compute-intensive** to train and deploy (cloud GPU resources)
- Poor at representing **uncertainty**
- **Easily fooled** by adversarial examples
- **Hard to optimise**: non-convex + choice of architecture, learning procedure, initialisation, etc, require expert knowledge and experimentation

LIMITATIONS OF DEEP LEARNING

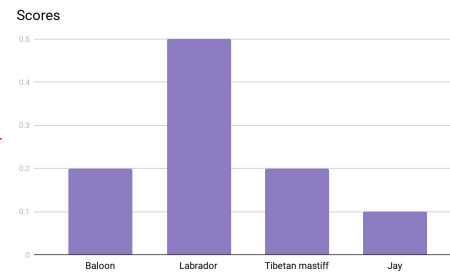
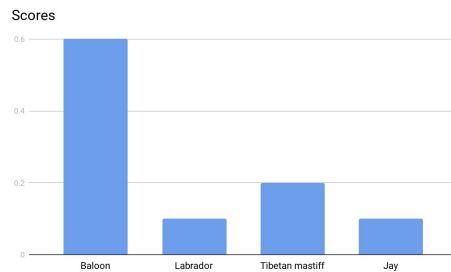
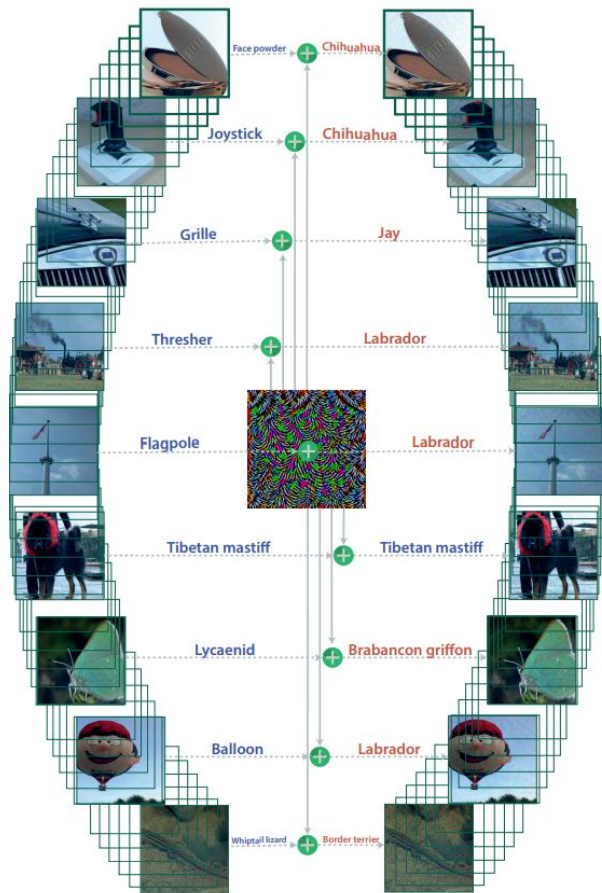
Neural networks and deep learning systems give amazing performance on many benchmark tasks, but they are generally:

- Uninterpretable **black-boxes**, lacking in transparency, difficult to trust
- Hard to perform reasoning with
- Assumed **single parameter** generated the data distribution
- Prone to **overfitting** (generalize poorly)
- Overly **confident** prediction about the input ($p(y|\mathbf{x}, \boldsymbol{\theta})$ is not the confidence!)

These networks can easily be fooled!

Adding a perturbation to images cause the natural images to be misclassified.

Moosavi-Dezfooli et al., CVPR 2017



Optimization

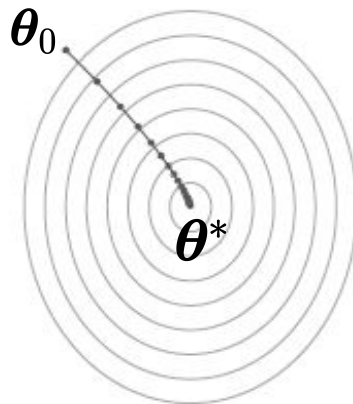
Gradient descent is the method we usually use to minimize the loss

It follows the gradient and updates the parameters

Gradient with respect to
the full training set

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} + \eta \frac{1}{n} \sum_{i=1}^n \nabla_{\boldsymbol{\theta}} \log(p(y_i | \mathbf{x}_i, \boldsymbol{\theta}_{t-1}))$$

Learning rate
(step size)

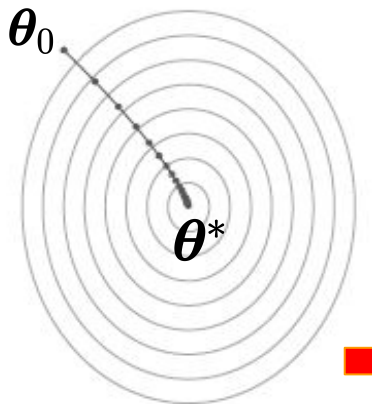


Optimization

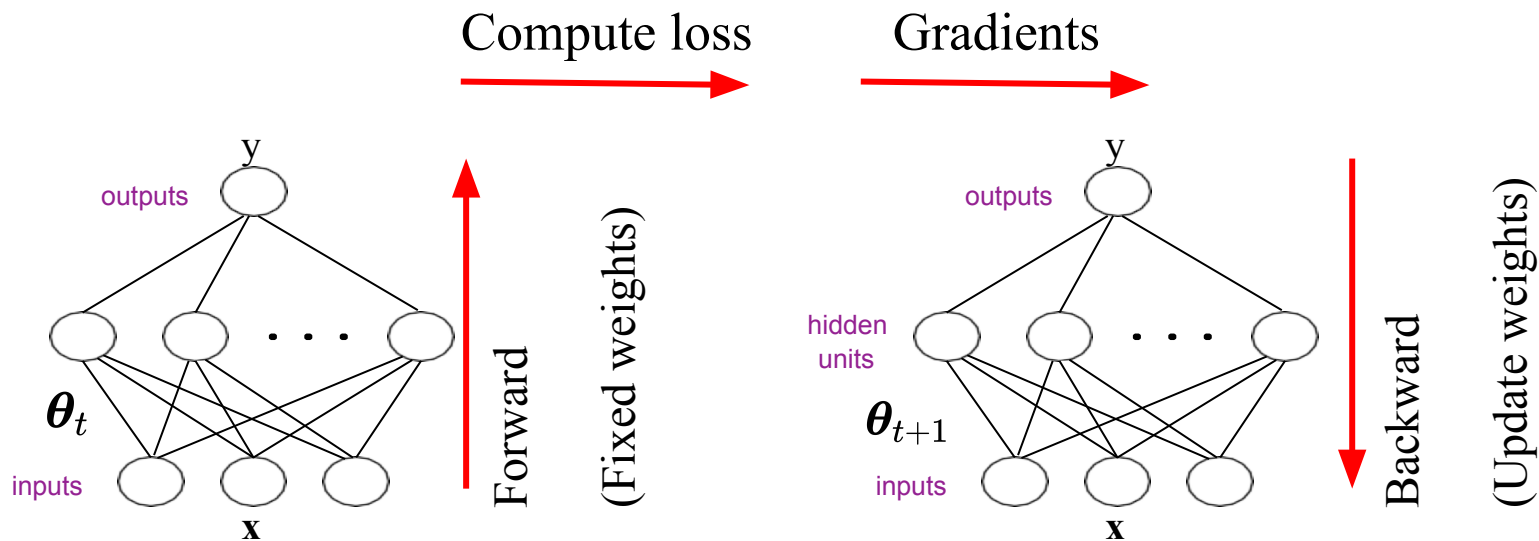
For large neural networks with a large training set, computing the gradient is costly.

Alternatively we “estimate” the full gradient with samples from the dataset

Called **stochastic gradient descent**

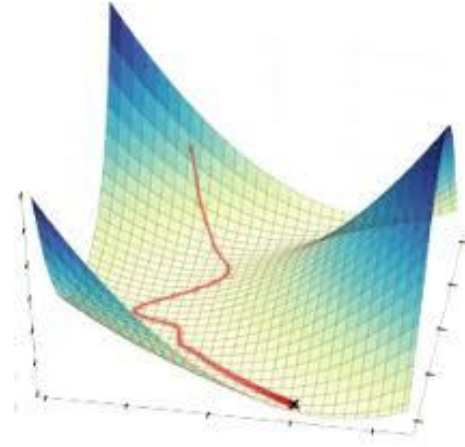
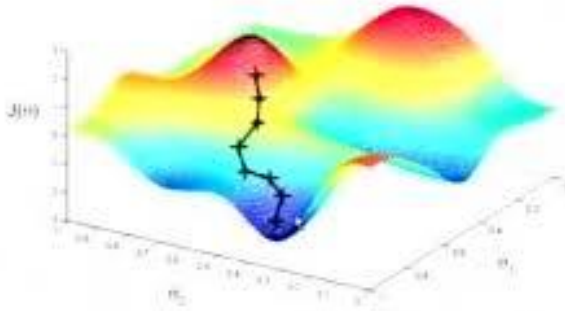


Updating the network



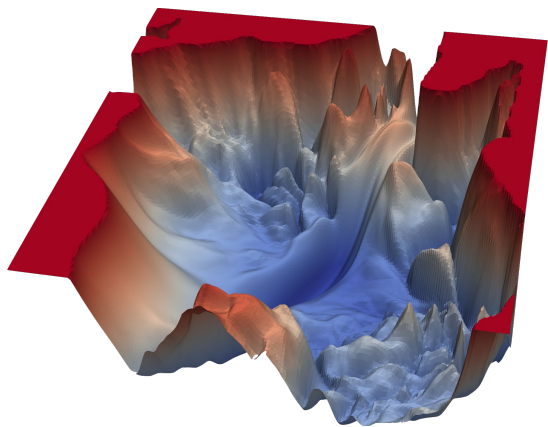
Using stochastic gradient descent is not ideal!

When minimizing the loss, we assume the landscape is smooth ...

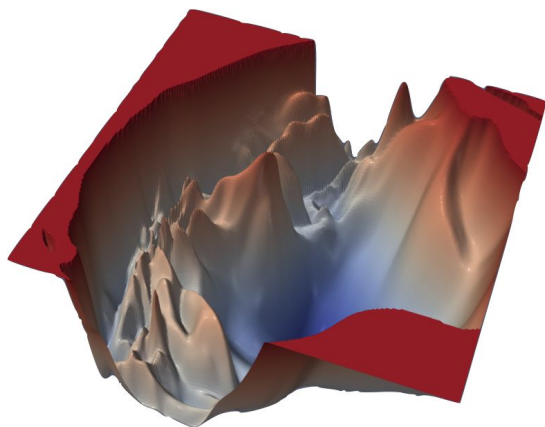


Using stochastic gradient descent is not ideal!

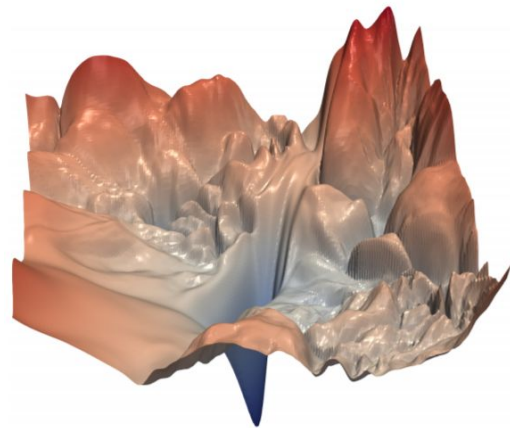
However the real loss landscape is



VGG-56



VGG-110



ResNet-56

Using stochastic gradient descent is not ideal!

Stochastic Gradient Descent (SGD) is a Terrible Optimizer

SGD is very noisy, and turns the gradient descent into a random walk over the loss.

However, it's very cheap, and cheap is all we can afford at scale.

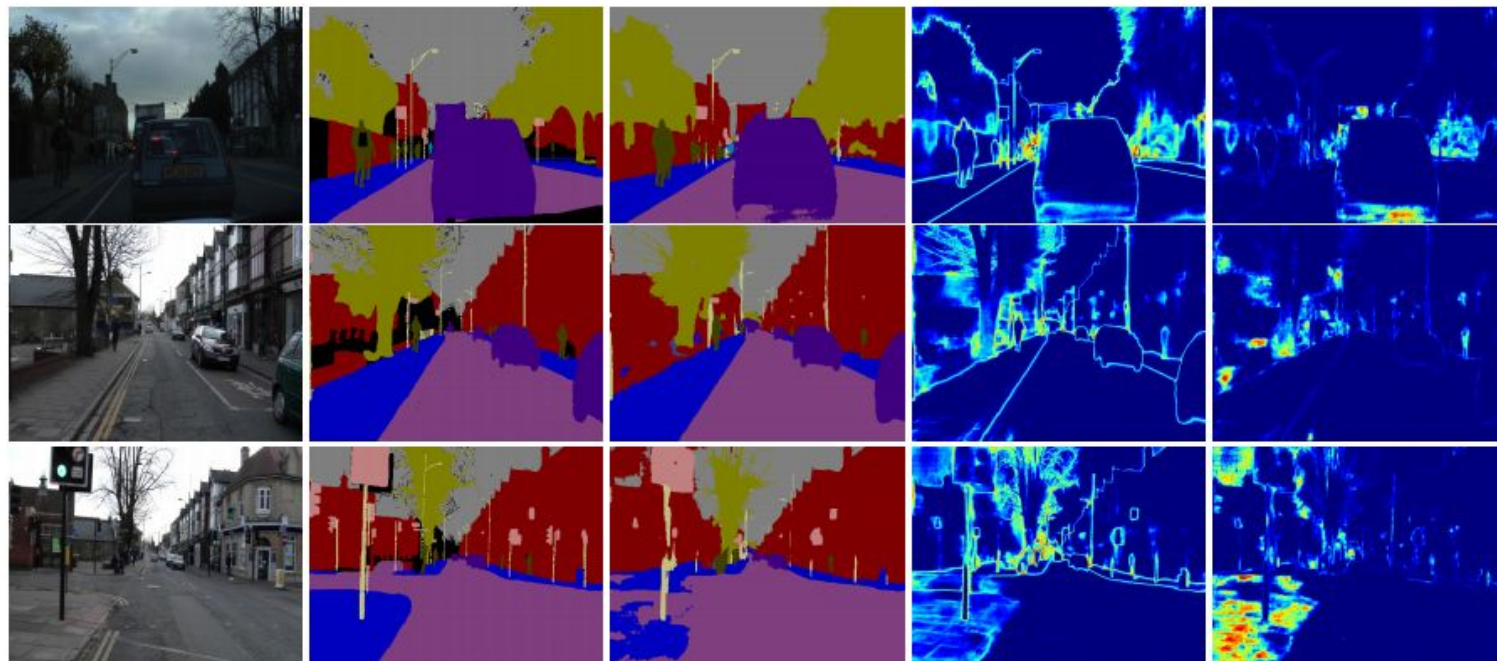
Types of Uncertainty

Aleatoric: Uncertainty inherent in the observation noise

Epistemic: Our ignorance about the correct model that generated the data

This includes the uncertainty in the model, parameters, convergence

Example



(a) Input Image

(b) Ground Truth

(c) Semantic Segmentation

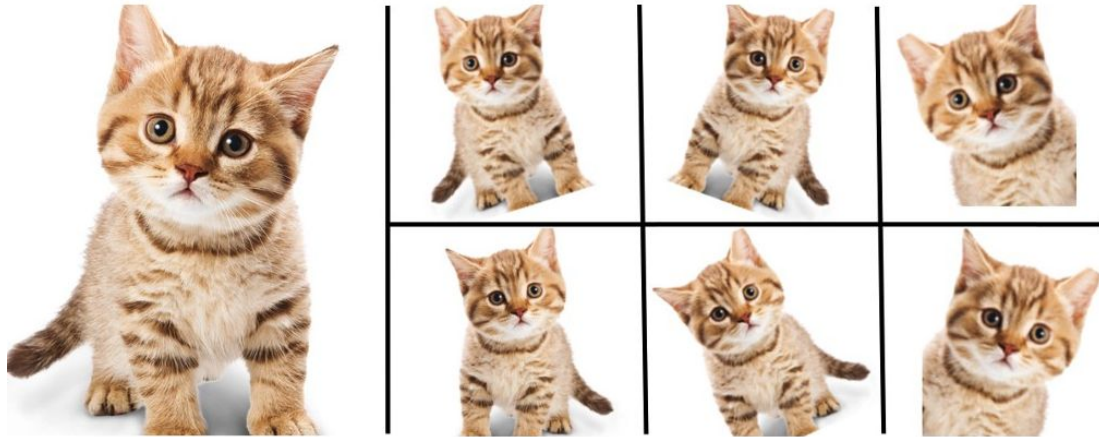
(d) Aleatoric Uncertainty

(e) Epistemic Uncertainty

Types of Uncertainty

Aleatoric: Uncertainty inherent in the observation noise

Data augmentation: add manipulated data to training set



Types of Uncertainty

Aleatoric: Uncertainty inherent in the observation noise

Ensemble methods, e.g.

1. Augment with adversarial training

$(\mathbf{x}_i + \Delta\mathbf{x}_i, y_i)$ where $\Delta\mathbf{x}_i = -\epsilon \text{sign}(\nabla_{\mathbf{x}} \log(p(y_i|\mathbf{x}_i)))$



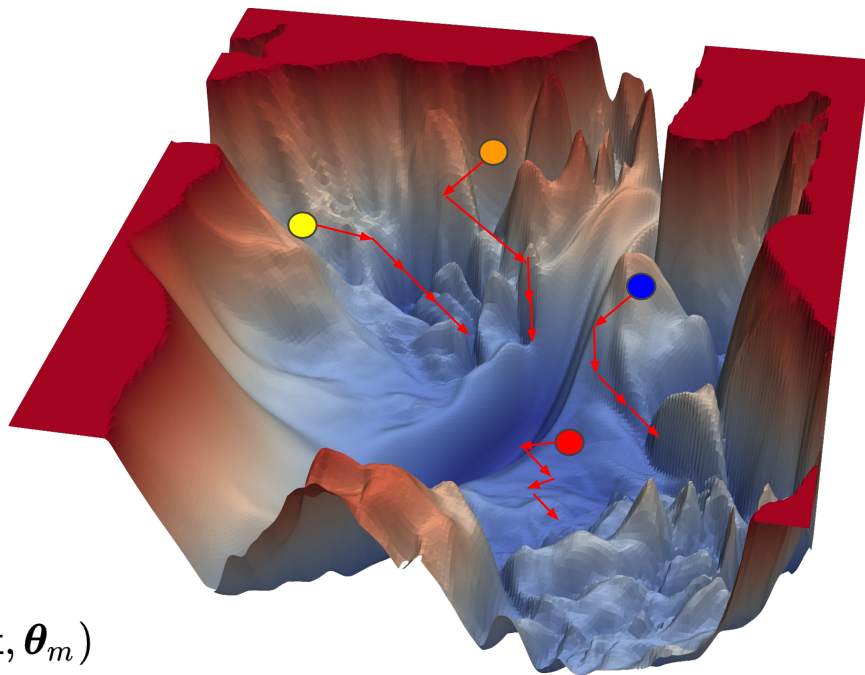
Minor change to the input

2. Encourage $p(y_i|\mathbf{x}_i, \theta)$ to be similar to $p(y_i|\mathbf{x}_i + \Delta\mathbf{x}_i, \theta)$
3. Train M models as an ensemble with random initialization
4. Combine at test for prediction $p(y|\mathbf{x}) = \frac{1}{M} \sum_{m=1}^M p(y|\mathbf{x}, \theta_m)$

Types of Uncertainty

Aleatoric: Uncertainty inherent in the observation noise

Ensemble methods



$$p(y|\mathbf{x}) = \frac{1}{M} \sum_{m=1}^M p(y|\mathbf{x}, \boldsymbol{\theta}_m)$$

Types of Uncertainty

Aleatoric: Uncertainty inherent in the observation noise

Epistemic: Our ignorance about the correct model that generated the data

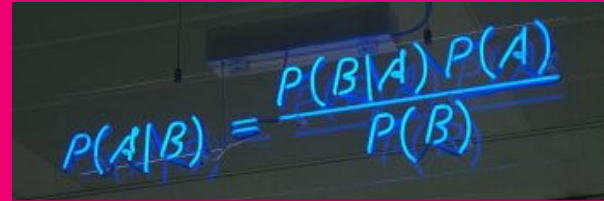
This includes the uncertainty in the model, parameters, convergence

Bayesian Methods

We can't tell what our model is certain about

We utilise Bayesian modeling ...

Bayesian Modeling addresses two uncertainties

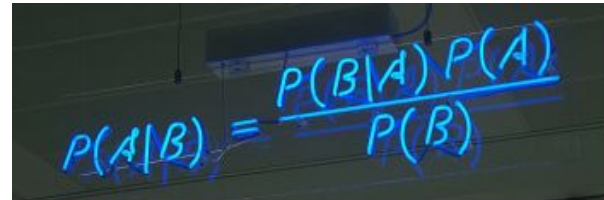
A photograph of a chalkboard with the Bayesian formula $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$ written in blue chalk. The chalkboard is set against a bright pink background.
$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

BAYES RULE

We have a prior about the world

We update our understanding of the world with the likelihood of events

We obtain a new belief about the world



A photograph of a chalkboard with the Bayes' Rule formula written in blue chalk. The formula is $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$. The chalkboard is dark, and the blue chalk is clearly visible against the background.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

BAYES RULE

$$P(\text{hypothesis} | \text{data}) = \frac{P(\text{hypothesis})P(\text{data} | \text{hypothesis})}{\sum_{\mathbf{h}} P(\mathbf{h})P(\text{data} | \mathbf{h})}$$

- Bayes rule tells us how to do inference about **hypotheses (uncertain quantities)** from **data (measured quantities)**.
- Learning and prediction can be seen as forms of inference.



Reverend Thomas Bayes (1702-1761)

BAYES RULE

$$P(\text{parameter} | \text{data}) = \frac{P(\text{parameter})P(\text{data} | \text{parameter})}{\underbrace{\sum_{\theta} P(\theta)P(\text{data} | \theta)}_{P(\text{data})}}$$

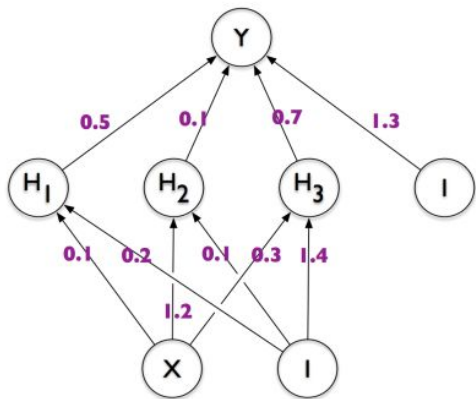
- Bayes rule tells us how to do inference about **hypotheses (uncertain quantities)** from **data (measured quantities)**.
- Learning and prediction can be seen as forms of inference.



Reverend Thomas Bayes (1702-1761)

Point Estimate of Neural Nets: Maximum A-posteriori Estimate (MAP)

- The weights are obtained by minimizing the expected loss
- It assumes the state of the world is fully realized by the mode of the distribution of the parameters



$$\max_{\theta} \log(p(\theta|\mathcal{D}))$$
$$\approx \frac{1}{n} \sum_{i=1}^n \log(p(y_i|\mathbf{x}_i, \theta)) + \log(p(\theta))$$

Regularizer

MAP is not Bayesian!

MAP is still a point estimate

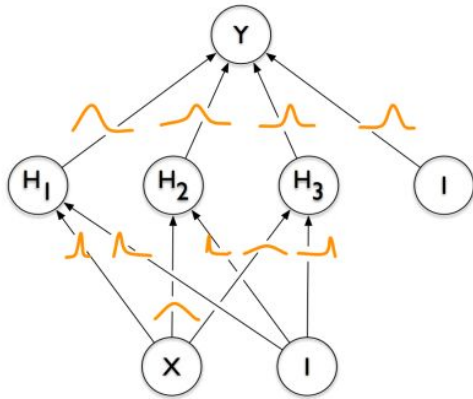
There is no distribution for the parameters

There is still a chance to reach a bad mode

WHAT DO I MEAN BY BEING BAYESIAN?

Dealing with all sources of **parameter uncertainty**

Also potentially dealing with **structure uncertainty**



Parameters θ are weights of neural net. They are assumed to be random variables.

Structure is the choice of architecture, number of hidden units and layers, choice of activation functions, etc.

Rules for Bayesian Machine Learning

Everything follows from two simple rules:

Sum rule: $P(x) = \sum_y P(x, y)$

Product rule: $P(x, y) = P(x)P(y|x)$

How Bayesians work

- Now, the distribution over parameters is (**Learning**),

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

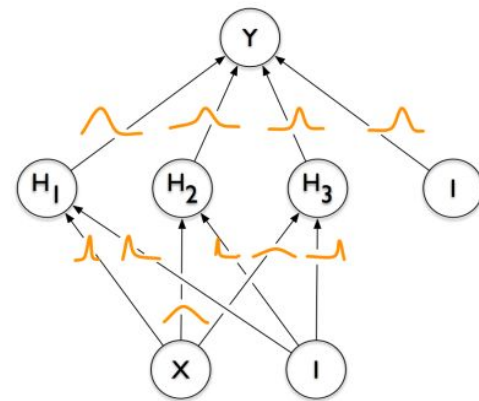
Posterior (circled in red) is the result of Likelihood (circled in red) multiplied by Prior (circled in red).

- The likelihood is

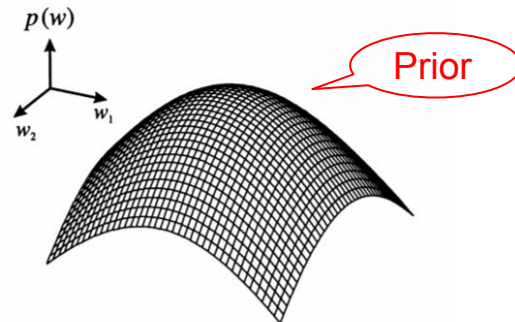
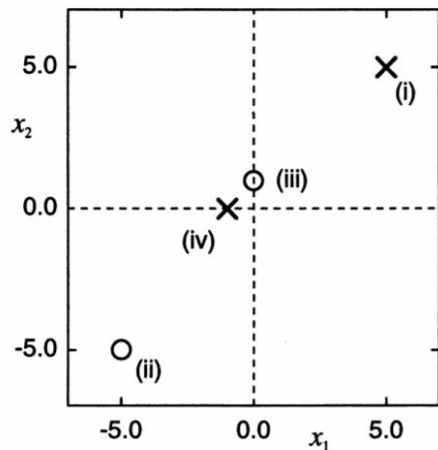
$$p(\mathcal{D}|\boldsymbol{\theta}) = \prod_{i=1}^n p(y_i|\mathbf{x}_i, \boldsymbol{\theta})$$

- There is an ideal test (**predictive**) distribution

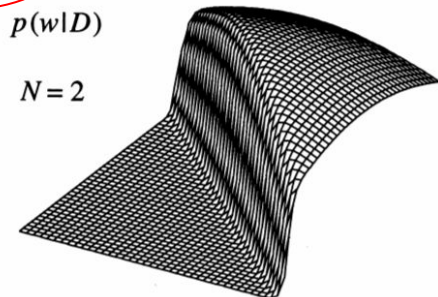
$$p(y^*|\mathbf{x}^*) = \int p(y^*|\mathbf{x}^*, \boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{D})d\boldsymbol{\theta}$$



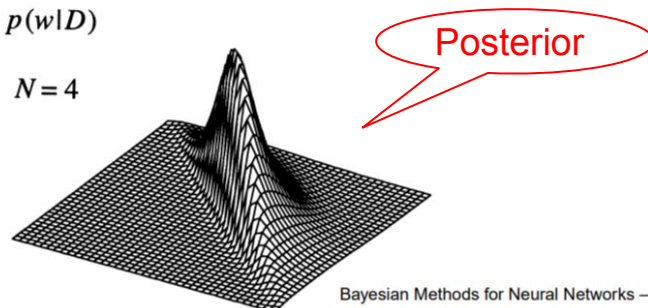
How Bayesians work



Likelihood



$p(w|D)$
 $N=4$



Inference using Sampling

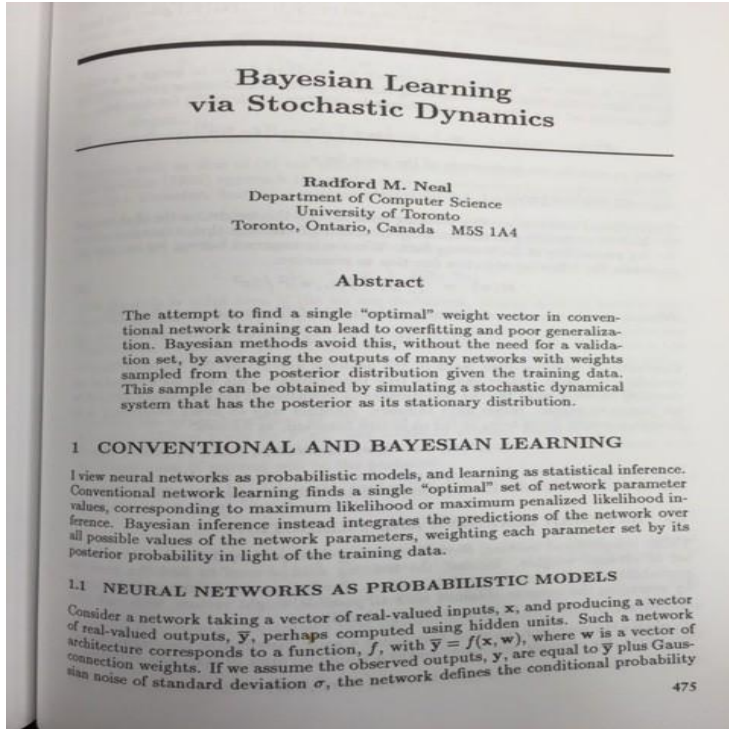
Computing $p(\boldsymbol{\theta}|\mathcal{D})$ is difficult. We can use

- Sampling (Monte Carlo methods)

$$\begin{aligned} p(\mathcal{D}) &= \int p(\boldsymbol{\theta})p(\mathcal{D}|\boldsymbol{\theta})d\boldsymbol{\theta} \\ &\approx \frac{1}{n} \sum_i p(\mathcal{D}|\boldsymbol{\theta}_i) \quad \boldsymbol{\theta}_i \sim p(\boldsymbol{\theta}) \end{aligned}$$

Variants of sampling methods: Gibbs, Metropolis-hastings,
Gradient-based ...

Long history



Neal, R.M. Bayesian learning via stochastic dynamics. In NIPS 1993.

First Markov Chain Monte Carlo (MCMC) sampling algorithm for Bayesian neural networks. Uses Hamiltonian Monte Carlo (HMC), a sophisticated MCMC algorithm that makes use of *gradients* to sample efficiently.

Langevin Dynamics

We said SGD randomly traverses the weight distributions. Let's utilise it

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} + \frac{\eta_t}{2} \left(\nabla_{\boldsymbol{\theta}} p(\boldsymbol{\theta}) + \frac{1}{n} \sum_{i=1}^n \nabla_{\boldsymbol{\theta}} \log(p(y_i | \mathbf{x}_i, \boldsymbol{\theta}_{t-1})) \right) + \boldsymbol{\epsilon}_t$$
$$\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \eta_t \mathbf{I})$$

Update SGD with a Gaussian noise

Learning rate decreases to zero

Langevin Dynamics

We said SGD randomly traverses the weight distributions. Let's utilise it

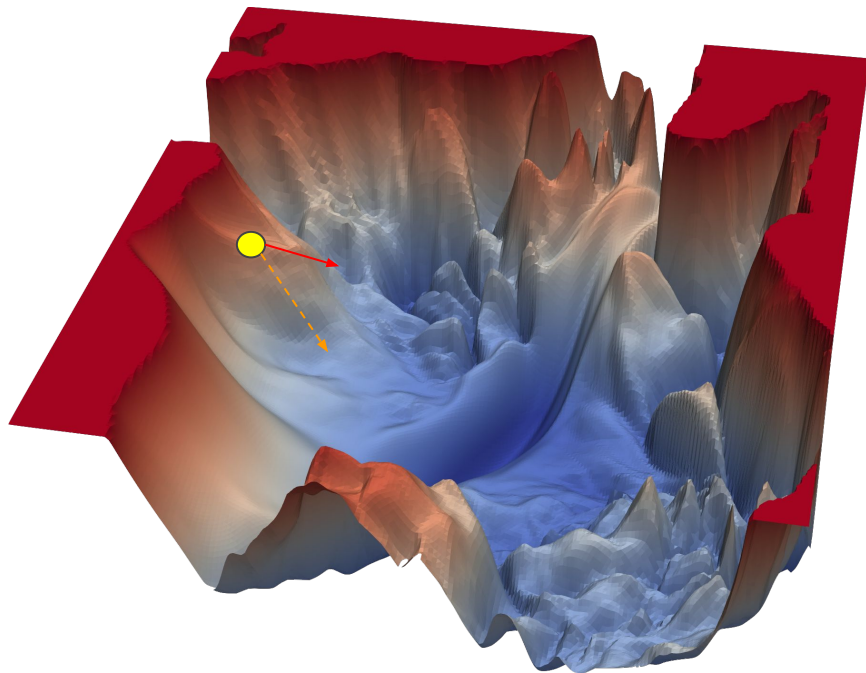
$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} + \frac{\eta_t}{2} (\nabla_{\boldsymbol{\theta}} p(\boldsymbol{\theta}) + \frac{1}{n} \sum_{i=1}^n \nabla_{\boldsymbol{\theta}} \log(p(y_i | \mathbf{x}_i, \boldsymbol{\theta}_{t-1}))) + \epsilon_t$$
$$\epsilon_t \sim \mathcal{N}(0, \eta_t \mathbf{I})$$

- Gradient term encourages dynamics to spend more time in high probability areas.
- Brownian motion provides noise so that dynamics will explore the whole parameter space.

Langevin Dynamics

Treat parameters of the network as different functions of the ensemble

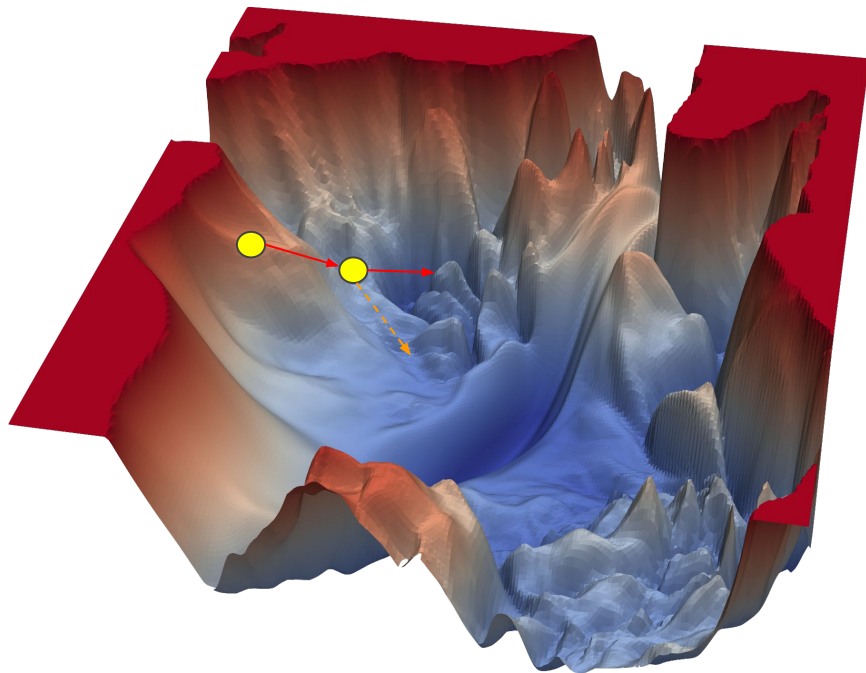
$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} + \frac{\eta_t}{2} (\nabla_{\boldsymbol{\theta}} p(\boldsymbol{\theta}) + \frac{1}{n} \sum_{i=1}^n \nabla_{\boldsymbol{\theta}} \log(p(y_i | \mathbf{x}_i, \boldsymbol{\theta}_{t-1}))) + \boldsymbol{\epsilon}_t$$
$$\boldsymbol{\epsilon}_t \sim \mathcal{N}(0, \eta_t \mathbf{I})$$



Langevin Dynamics

Treat parameters of the network as different functions of the ensemble

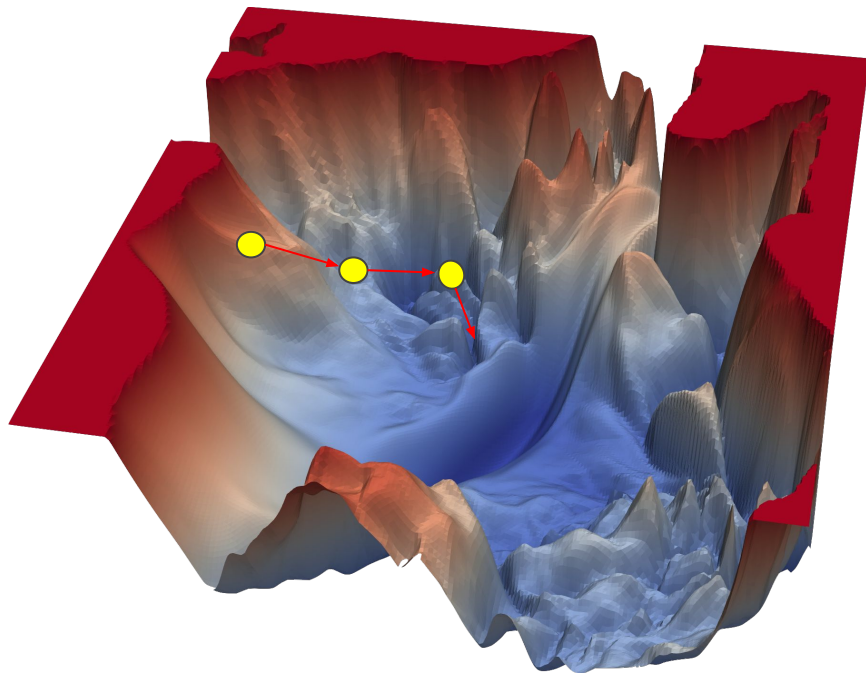
$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} + \frac{\eta_t}{2} (\nabla_{\boldsymbol{\theta}} p(\boldsymbol{\theta}) + \frac{1}{n} \sum_{i=1}^n \nabla_{\boldsymbol{\theta}} \log(p(y_i | \mathbf{x}_i, \boldsymbol{\theta}_{t-1}))) + \boldsymbol{\epsilon}_t$$
$$\boldsymbol{\epsilon}_t \sim \mathcal{N}(0, \eta_t \mathbf{I})$$



Langevin Dynamics

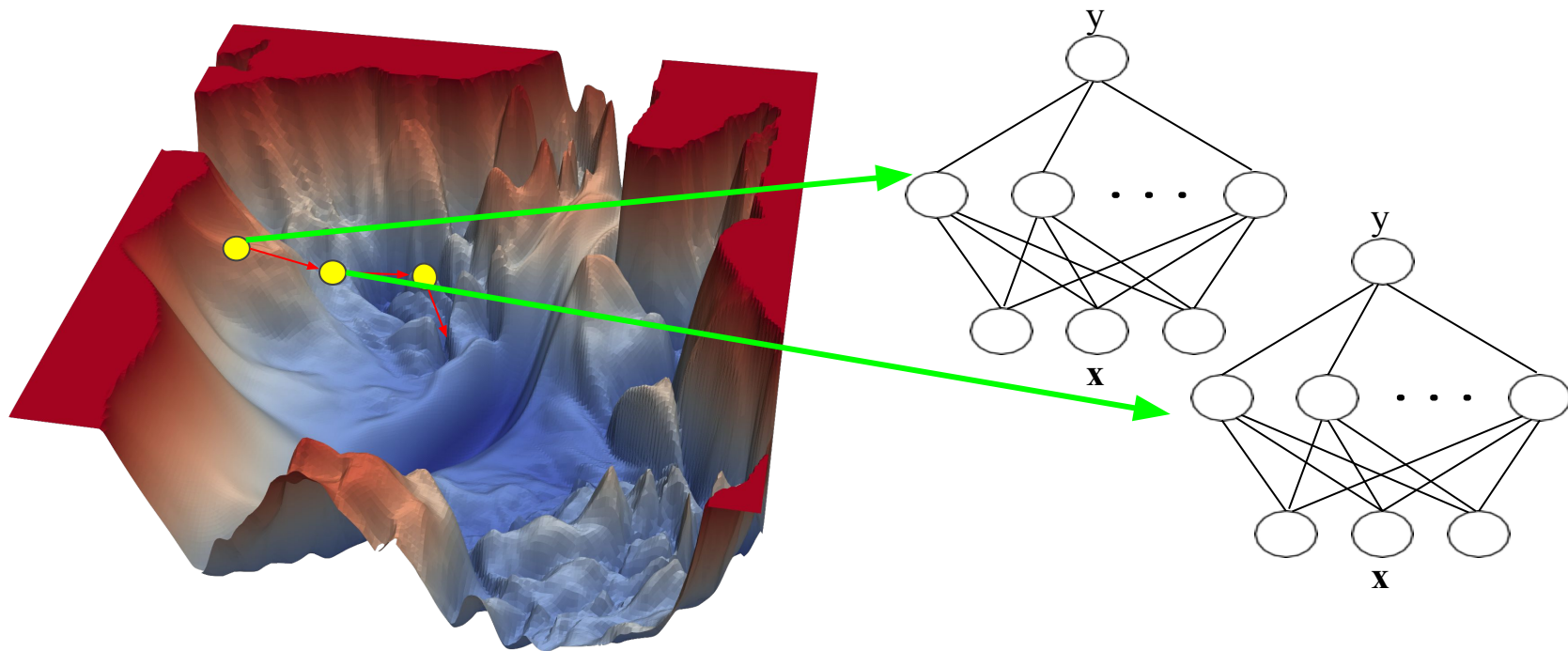
Treat parameters of the network as different functions of the ensemble

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} + \frac{\eta_t}{2} (\nabla_{\boldsymbol{\theta}} p(\boldsymbol{\theta}) + \frac{1}{n} \sum_{i=1}^n \nabla_{\boldsymbol{\theta}} \log(p(y_i | \mathbf{x}_i, \boldsymbol{\theta}_{t-1}))) + \epsilon_t$$
$$\epsilon_t \sim \mathcal{N}(0, \eta_t \mathbf{I})$$



Langevin Dynamics

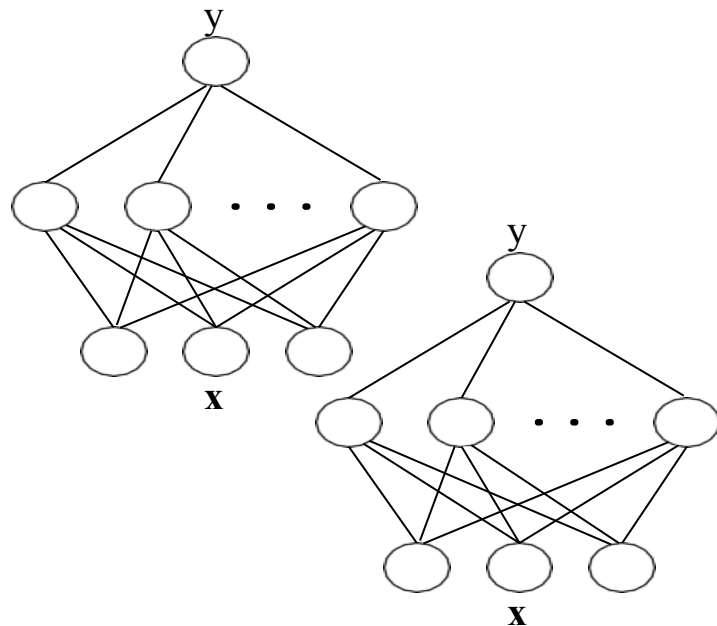
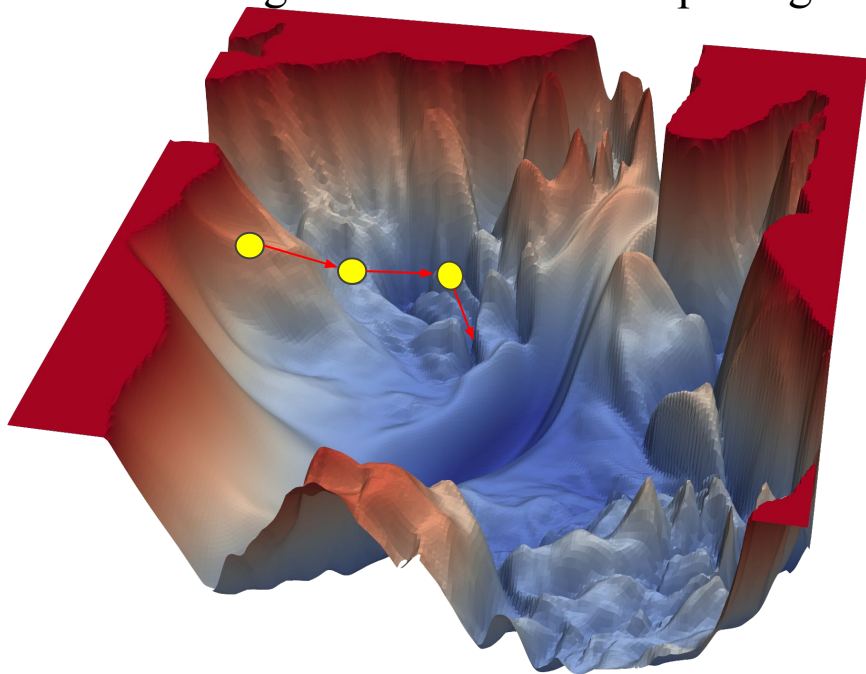
Each parameter sample generates a new network



Langevin Dynamics

There would be a lot more networks from the low loss regions

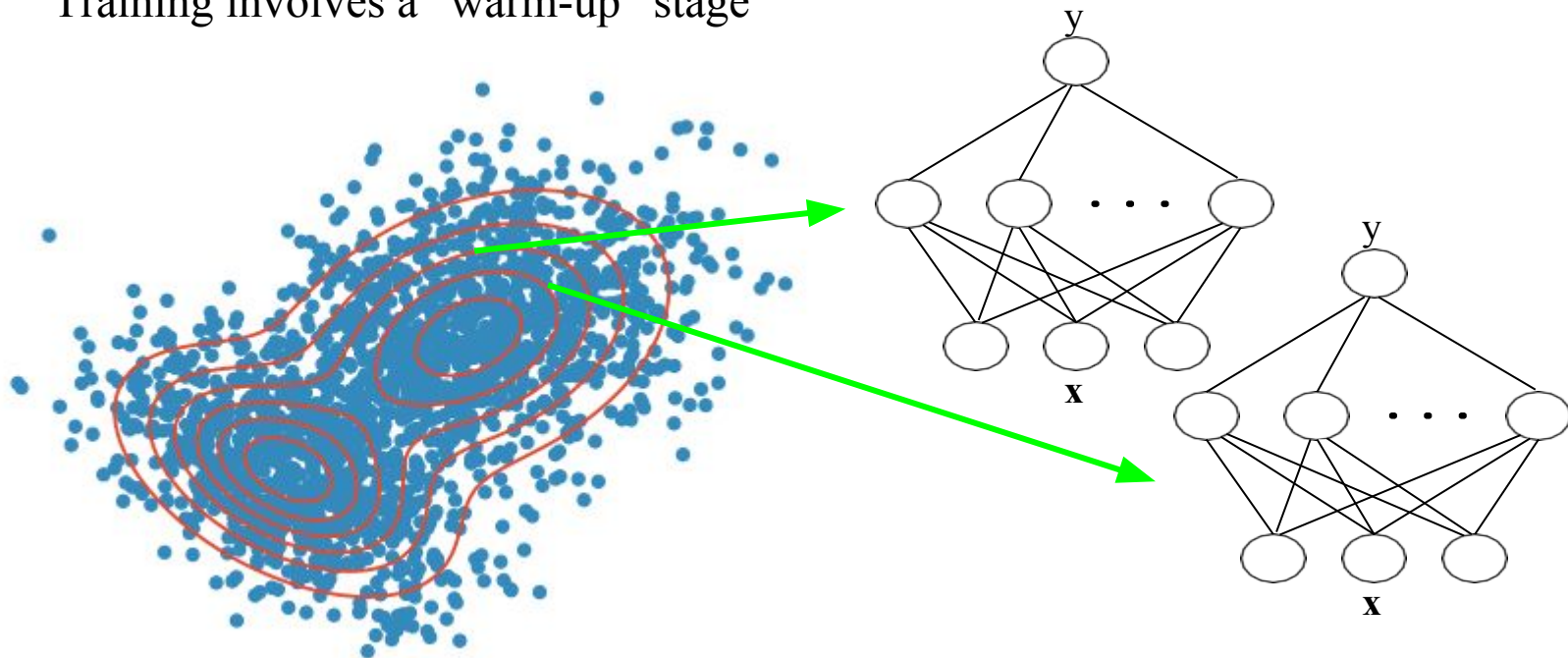
Training involves a “warm-up” stage



Langevin Dynamics

There would be a lot more networks from the low loss regions

Training involves a “warm-up” stage



Why not

- Is desirable for smaller dimensional problems
- Sampling methods are computationally demanding
- Sometimes requires a large sample size to perform well
- Theoretically unbiased estimates, however in practice is biased

Approximate Inference

Computing $p(\boldsymbol{\theta}|\mathcal{D})$ is difficult. We can use

- Approximate with an alternative simpler distribution

$$q_{\omega}(\boldsymbol{\theta}) \approx p(\boldsymbol{\theta}|\mathcal{D})$$

Then the predictive distribution is

$$p(y^*|\mathbf{x}^*) = \int p(y^*|\mathbf{x}^*, \boldsymbol{\theta})q_{\omega}(\boldsymbol{\theta})d\boldsymbol{\theta}$$

Integrate out the parameters; otherwise sample

Approximate Inference to optimize alternative distribution

Computing $p(\boldsymbol{\theta}|\mathcal{D})$ is difficult. We can use

- Minimize the divergence with respect to ω

$$\text{KL}(q_{\omega}(\boldsymbol{\theta})||p(\boldsymbol{\theta}|\mathcal{D}))$$

- Identical to minimizing

Entropy of the data

Prior

$$\mathcal{L}(\omega) = - \int \log(p(\mathcal{D}|\boldsymbol{\theta}))q_{\omega}(\boldsymbol{\theta})d\boldsymbol{\theta} + \text{KL}(q_{\omega}(\boldsymbol{\theta})||p(\boldsymbol{\theta}))$$

The inference is now cast to an optimization problem.

Approximate Inference to optimize alternative distribution

- Using Monte Carlo sampling, we can rewrite the integration

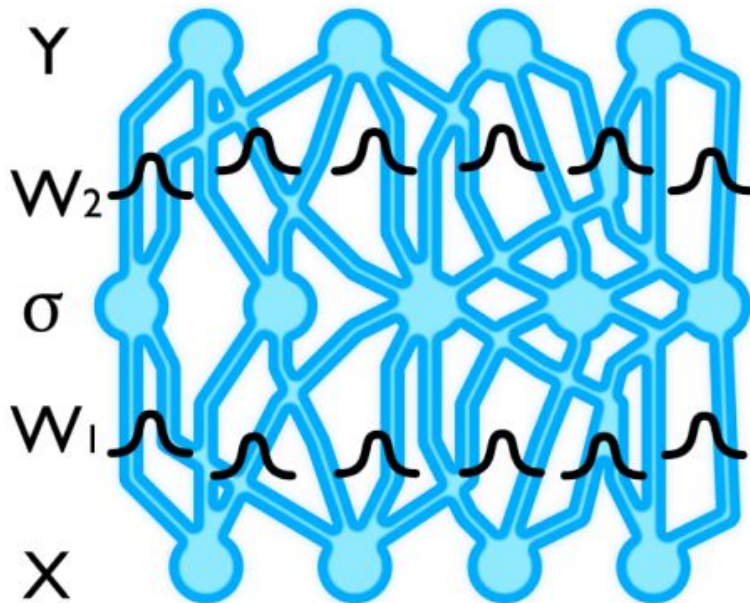
$$\mathcal{L}(\omega) = - \int \log(p(\mathcal{D}|\boldsymbol{\theta}))q_{\omega}(\boldsymbol{\theta})d\boldsymbol{\theta} + \text{KL}(q_{\omega}(\boldsymbol{\theta})||p(\boldsymbol{\theta}))$$

As an unbiased estimate of the integral (for one sample)

$$\hat{\mathcal{L}}(\omega) = - \log(p(\mathcal{D}|\boldsymbol{\theta}))q_{\omega}(\boldsymbol{\theta}) + \text{KL}(q_{\omega}(\boldsymbol{\theta})||p(\boldsymbol{\theta}))$$

Approximate Inference for Bayesian NNs

For Neural networks, it is hard to find the posterior. We need a distribution for each weight.



Approximate Inference for Bayesian NNs

For Neural networks, it is hard to find the posterior.

One possible solution is to use the following alternative distribution q

- Assume all the parameters in layers are independent
- The alternative distribution is a mixture model

$$q_{\omega_{i,k}}(\boldsymbol{\theta}_{i,k}) = \eta\delta(\boldsymbol{\theta}_{i,k} - 0) + (1 - \eta)\delta(\boldsymbol{\theta}_{i,k} - \omega_{i,k})$$

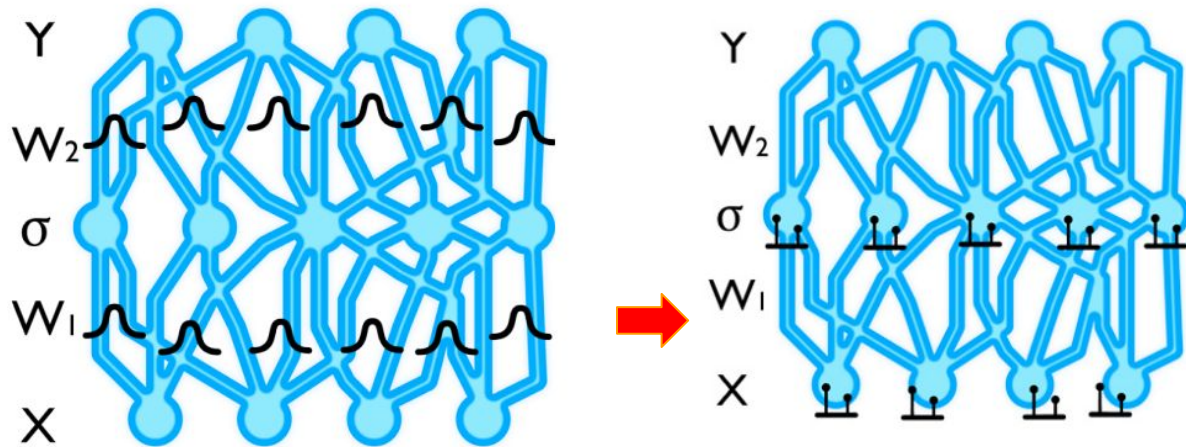
For k 's component of the i 'th layer

Approximate Inference for Bayesian NNs

Now, minimising the variational loss,

$$\hat{\mathcal{L}}(\omega) = -\log(p(\mathcal{D}|\theta))q_{\omega}(\theta)d\theta + \text{KL}(q_{\omega}(\theta)||p(\theta))$$

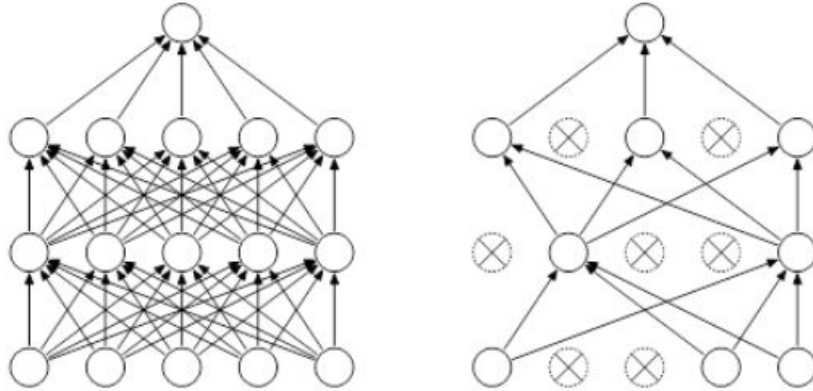
is equal to dropping units in the network



Dropout

Set 50% of the activations to zero randomly.

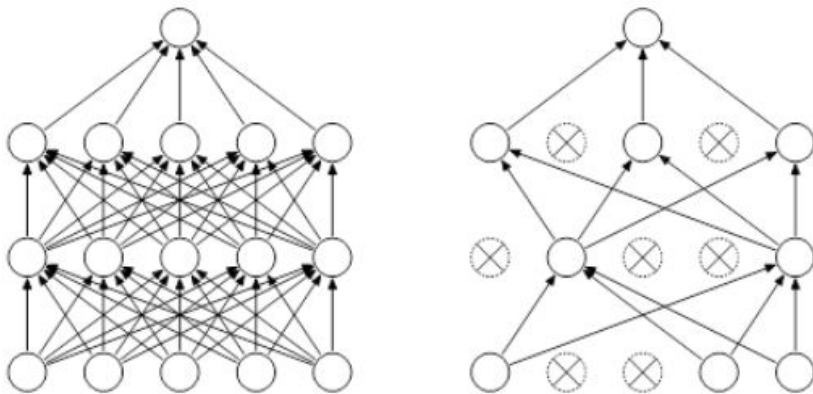
- Force other parts of the network to learn redundant representations.
- Sounds crazy, but works great.



Dropout

In test time, we take “samples” from the network and average the output

The variance of the outputs is the confidence in prediction

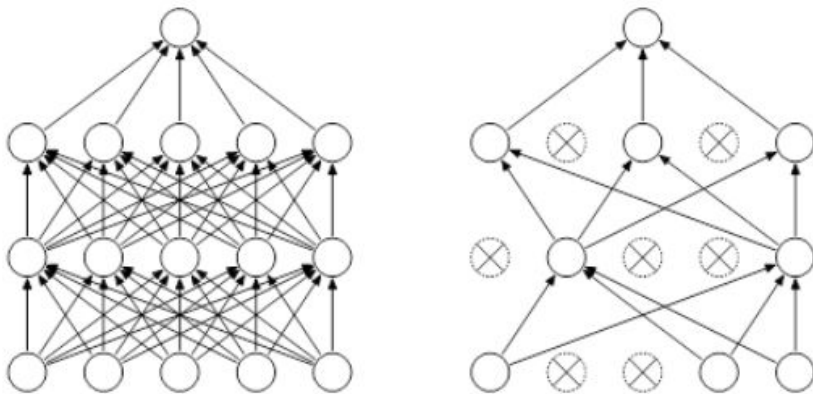


Dropout

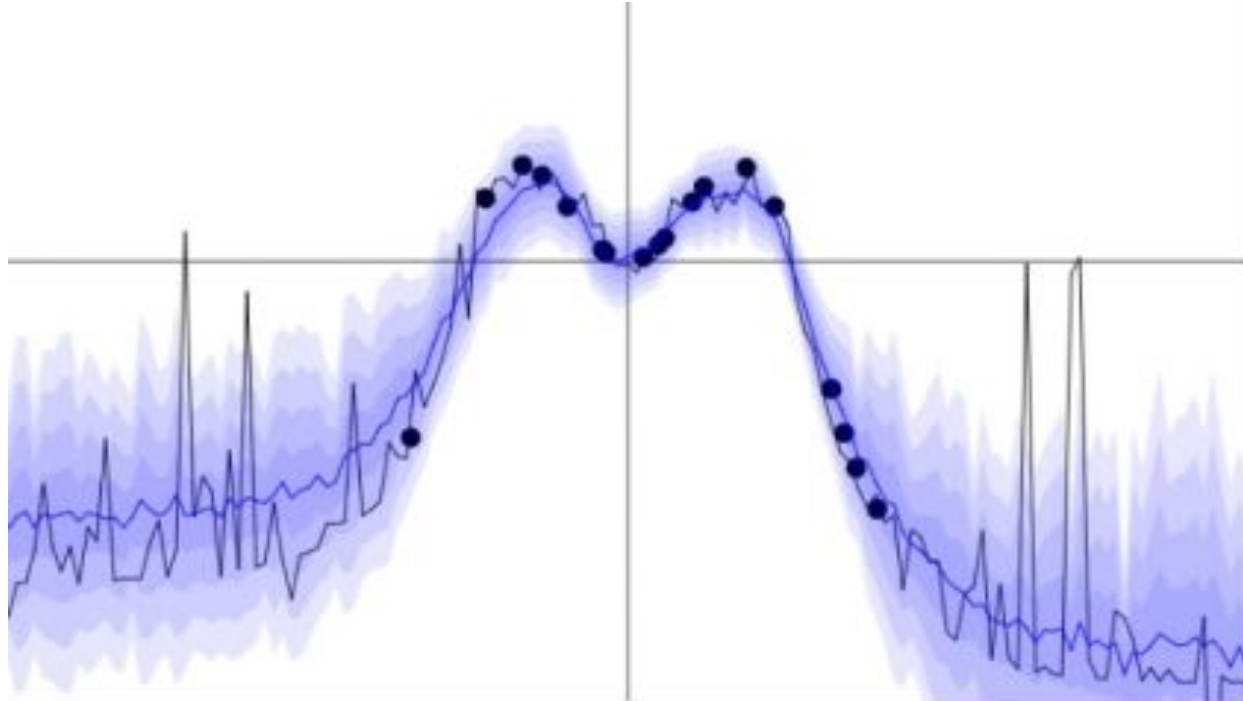
The alternative distribution is a set of Bernoulli (cheap multi-modality)

It is cheap to evaluate

Easy to implement and requires minimum change to the network structure



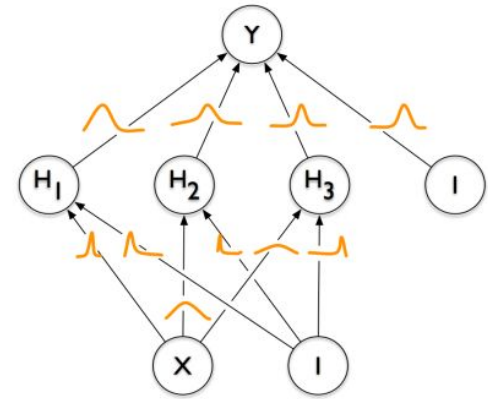
Dropout for Bayesian NN



Alternative

Assume the weights are Gaussian ...

Again use gradient with respect to parameters to update



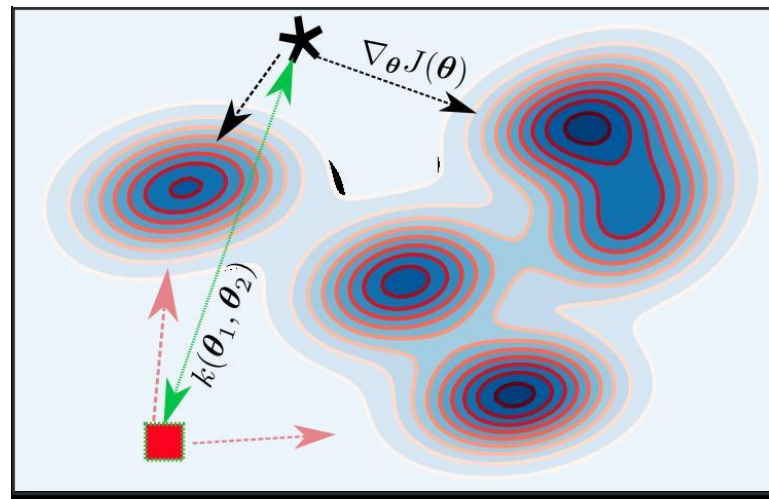
Stein Variational Inference

Keep particles (small sample size of parameters)

Start from initial point

Update in parallel with interactions

$$\hat{\psi}(\theta_i) = \frac{1}{n} \sum_{j=1}^n [\nabla_{\theta_j} \log \pi(\theta_j | a^*, q^{(t)}, C, I) k(\theta_j, \theta_i) + \nabla_{\theta_j} k(\theta_j, \theta_i)].$$



WHY SHOULD WE CARE?

Calibrated model and prediction uncertainty: getting systems that know when they don't know.

Automatic model **complexity control and structure learning**
(Bayesian Occam's Razor)

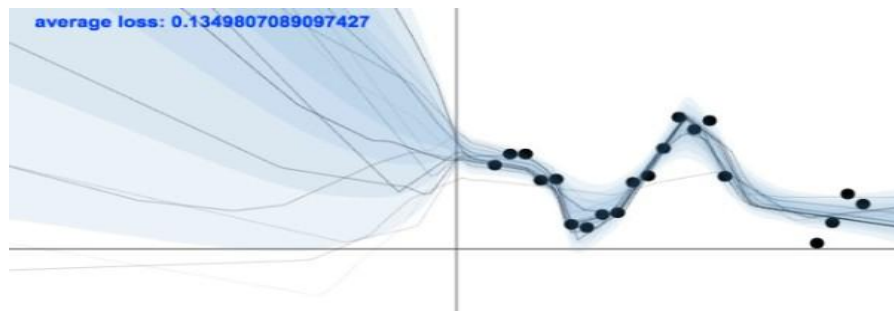


Figure from Yarin Gal's thesis "Uncertainty in Deep Learning" (2016)

Bayesian ...

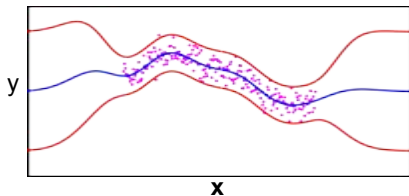
- It is self-regularized (there is the average of parameters involved)
- We have a distribution of the parameters
- **Uncertainty** is estimated for free
- Both uncertainty types are handled
- Prior knowledge is easily incorporated

However ...

- It is computationally demanding
- The integrals are intractable
- Parameters are high dimensional

GAUSSIAN PROCESSES

Consider the problem of **nonlinear regression**: You want to learn a **function** f with **error bars** from data $D = \{\mathbf{X}, \mathbf{y}\}$



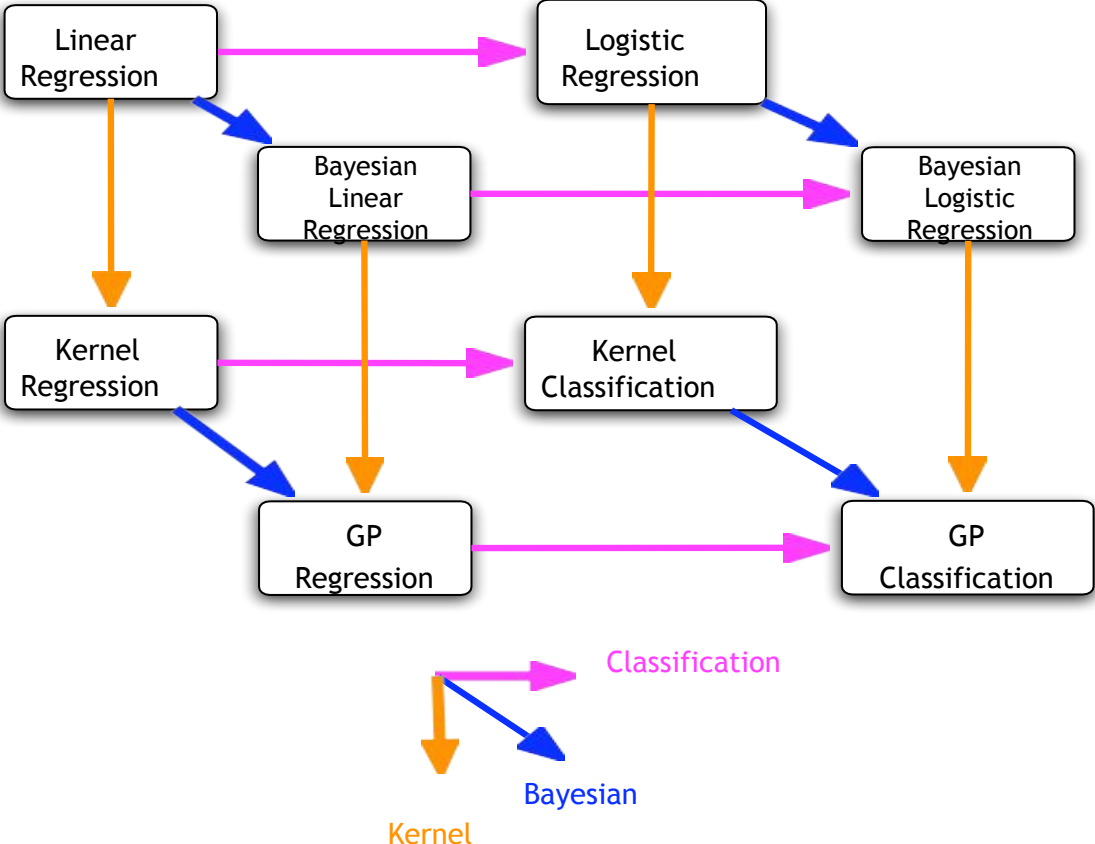
A **Gaussian process** defines a distribution over functions $p(f)$ which can be used for Bayesian regression:

$$p(f | D) = \frac{p(f)p(D|f)}{p(D)}$$

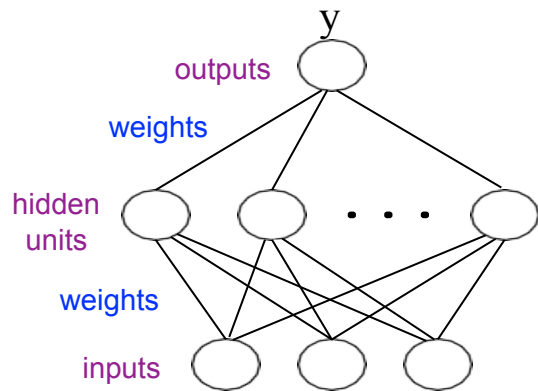
Definition: $p(f)$ is a **Gaussian process** if for *any* finite subset $\{x_1, \dots, x_n\} \subset X$, the marginal distribution over that subset $p(\mathbf{f})$ is multivariate Gaussian.

GPs can be used for regression, classification, ranking, dim. reduct...

A PICTURE: GPs, LINEAR AND LOGISTIC REGRESSION, AND SVMs



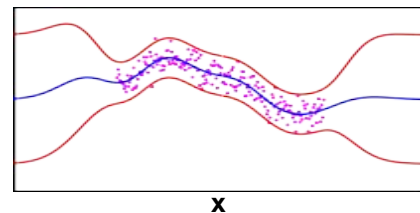
NEURAL NETWORKS AND GAUSSIAN PROCESSES



Bayesian neural network

\mathbf{x}

A neural network with one hidden layer, infinitely many hidden units and Gaussian priors on the weights is a Gaussian process (Neal, 1994)



What else?

Laplace approximation

Bayesian Information Criterion (BIC)

Variational approximations

Expectation Propagation (EP)

Markov chain Monte Carlo methods (MCMC)

Sequential Monte Carlo (SMC)

Exact Sampling

CONCLUSIONS

Probabilistic modelling offers a general framework for building systems that **learn from data**

Advantages include better estimates of **uncertainty**, automatic ways of **learning structure** and **avoiding overfitting**, and a principled foundation.

Disadvantages include higher **computational cost**, depending on the approximate inference algorithm

Bayesian neural networks have a long history and are undergoing a tremendous wave of revival.