Lecture 9 - *Monte Carlo*

OBJECTIVE: Monte Carlo techniques are used to carry out integration, simulation and optimisation in large dimensional spaces. They allow us to carry out inference and learning with complex intractable models. In this lecture, we will learn about Monte Carlo, importance sampling, Markov chain Monte Carlo (MCMC) and particle filters.

◇ MONTE CARLO

Monte Carlo methods enable us to solve the following problems:

1. *Normalisation*: To obtain the posterior $p(x|y)$ given the prior $p(x)$ and likelihood $p(y|x)$, the normalising factor in Bayes' theorem needs to be computed

$$p(x|y) = \frac{p(y|x)p(x)}{\int_{\mathcal{X}} p(y|x')p(x')dx'}.$$

2. *Marginalisation*: Given the joint posterior of $(x, z) \in$

$\mathcal{X} \times \mathcal{Z}$, we may often be interested in the marginal posterior

$$p(x|y) = \int_{\mathcal{Z}} p(x, z|y)dz.$$

3. *Expectation*: The objective of the analysis is often to obtain summary statistics of the form

$$\mathbb{E}_{p(x|y)}(f(x)) = \int_{\mathcal{X}} f(x)p(x|y)dx$$

for some function of interest $f : \mathcal{X} \to \mathbb{R}^{n_f}$ integrable with respect to $p(x|y)$. Examples of appropriate functions include the conditional mean, in which case $f(x) = x$, or the conditional covariance of x where $f(x) = xx' - \mathbb{E}_{p(x|y)}(x)\mathbb{E}'_{p(x|y)}(x)$.

4. **Statistical mechanics**: Here, one needs to compute the partition function Z of a system with states s and Hamiltonian $E(s)$

$$Z = \sum_s \exp\left[-\frac{E(s)}{kT}\right],$$

where k is the Boltzmann's constant and T denotes the temperature of the system. Summing over the large number of possible configurations is prohibitively expensive. Note that the problems of computing the partition function and the normalising constant in statistical inference are analogous.

5. **Optimisation**: The goal of optimisation is to extract the solution that minimises some objective function from a large set of feasible solutions. In fact, this set can be continuous and unbounded. In general, it is too computationally expensive to compare all the solutions to find out which one is optimal.

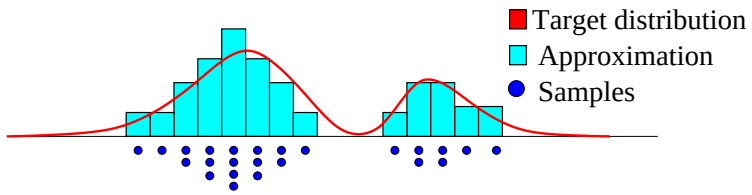
6. **Simulation**: One often needs to simulate physical systems in physics and computer graphics.

The Monte Carlo Principle

The idea of Monte Carlo simulation is to draw an i.i.d. set of samples $\{x^{(i)}\}_{i=1}^{N}$ from a target density $p(x)$ defined on a high-dimensional space $\mathcal{X}$. These N samples can be used to approximate the target distribution with the following empirical point-mass function (think of it as a histogram):

$$p_N(dx) = \frac{1}{N}\sum_{i=1}^{N} \delta_{x^{(i)}}(dx),$$

where $\delta_{x^{(i)}}(dx)$ denotes the delta-Dirac mass located at $x^{(i)}$.



Consequently, one can approximate the integrals (or very large sums) $I(f)$ with tractable sums $I_N(f)$ as follows

⋆

$$I(f) = \int_{\mathcal{X}} f(x)p(x)dx.$$

- The advantage of Monte Carlo integration over deterministic integration arises from the fact that the former positions the integration grid (samples) in regions of high probability.
- The N samples can also be used to obtain a maximum of the objective function $p(x)$ as follows

$$\widehat{x} = \underset{x^{(i)};i=1,\ldots,N}{\arg\max} \; p\left(x^{(i)}\right)$$

  However, we will show later that it is possible to construct simulated annealing algorithms that allow us to sample approximately from a distribution whose support is the set of global maxima.
- When $p(x)$ has standard form, *e.g.* Gaussian, it is straightforward to sample from it using easily available routines. However, when this is not the case, we need to introduce more sophisticated techniques based on importance sampling and MCMC.

Importance Sampling

Importance sampling is a "classical" solution that goes back to the 1940's. Let us introduce an arbitrary importance proposal distribution $q(x)$ such that its support includes the support of $p(x)$ and such that we can sample from it. Then we can rewrite $I(f)$ as follows

$$I(f) = \int f(x)\, w(x)\, q(x)\, dx$$

where $w(x) \triangleq \frac{p(x)}{q(x)}$ is known as the *importance weight*.

⋆ Proof:

Consequently, if one can simulate N i.i.d. samples $\{x^{(i)}\}_{i=1}^{N}$ according to $q(x)$ and evaluate $w(x^{(i)})$, a possible Monte Carlo estimate of $I(f)$ is

⋆

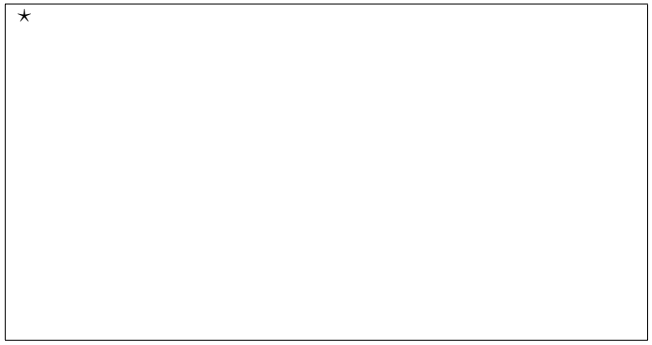
$\widehat{I}_N(f) =$

This estimator is unbiased and, under weak assumptions, the strong law of large numbers applies, that is $\widehat{I}_N(f) \xrightarrow[N\to\infty]{a.s.} I(f)$. It is clear that this integration method can also be interpreted as a sampling method where the posterior density $p(x)$ is approximated by:

$$\widehat{p}_N(dx) = \frac{1}{N}\sum_{i=1}^{N} w(x^{(i)})\delta_{x^{(i)}}(dx)$$

Some proposal distributions $q(x)$ will obviously be preferable to others.

When the normalising constant of $p(x)$ is unknown, it is still possible to apply the importance sampling method:



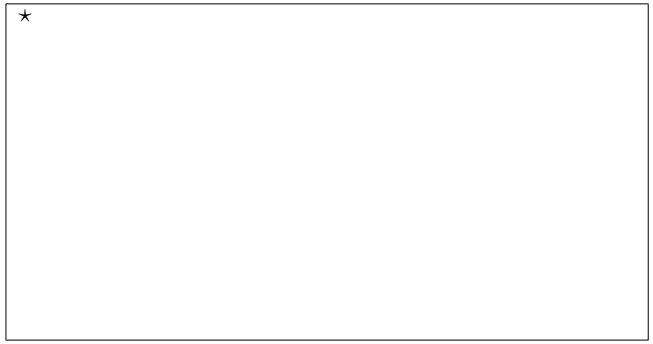
The Monte Carlo estimate of $I(f)$ becomes

$$\widetilde{I}_N(f) = \frac{\frac{1}{N}\sum_{i=1}^{N} f\left(x^{(i)}\right) w(x^{(i)})}{\frac{1}{N}\sum_{j=1}^{N} w\left(x^{(i)}\right)} = \sum_{i=1}^{N} f\left(x^{(i)}\right) \widetilde{w}(x^{(i)})$$

where $\widetilde{w}(x^{(i)})$ is a normalised importance weight. For N finite, $\widetilde{I}_N(f)$ is biased (ratio of two estimates) but asymptotically, under weak assumptions, the strong law of large numbers applies, that is $\widetilde{I}_N(f) \xrightarrow[N\to\infty]{a.s.} I(f)$.

Sampling Importance Resampling (SIR)

If one is interested in obtaining M *i.i.d.* samples from $\widehat{p}_N(x)$, then an asymptotically ($N/M \to \infty$) valid method consists of resampling M times according to the discrete distribution $\widehat{p}_N(x)$.



This procedure results in M samples $\widetilde{x}^{(i)}$ with the possibility that $\widetilde{x}^{(i)} = \widetilde{x}^{(j)}$ for $i \neq j$. After resampling, the approximation of the target density is

$$\widetilde{p}_M(dx) = \frac{1}{M}\sum_{i=1}^{M} \delta_{\widetilde{x}^{(i)}}(dx)$$

The SIR algorithm to sample from the posterior $p(x|y)$ is as follows:

Set $i = 1$

Repeat until $i = N$

1. Sample $x^{(i)} \sim q(x)$
2. Evaluate $p(x^{(i)}|y)$ up to a normalising constant.
3. Evaluate $q(x^{(i)})$ up to a normalising constant.
4. Compute $w(x^{(i)})$.

Normalise $w(x^{(i)})$ to obtain $\widetilde{w}(x^{(i)})$.

Resample $\{x^{(i)}, \widetilde{w}(x^{(i)})\}_{i=1}^{N} \longrightarrow \{\widetilde{x}^{(i)}, 1/N\}_{i=1}^{N}$

Example: Logistic Regression and Binary Classification

Given the input-output i.i.d. data sets $x \triangleq x_{1:T} \triangleq \{x_0, x_1, \ldots, x_T\}$ and $y \triangleq y_{1:T} \triangleq \{y_0, y_1, \ldots, y_T\}$, where $x_t \in \mathbb{R}$ and $y_t \in \{0, 1\}$. The idea is to come up with a model that takes a new input x_{T+1} and produces as output $p(y_{T+1} = 1|x_{T+1})$ and $p(y_{T+1} = 0|x_{T+1})$. This classification problem arises in several areas of technology, including condition monitoring and binary decision systems. For example, when monitoring patients, we might wish to decide whether they require an increase in drug intake based on new evidence.

For practical reasons, we parameterise our model. In particular, we introduce the following Bernoulli likelihood function:

$$p(y_t|x_t,\theta) = \left[\frac{1}{1+\exp(-\theta x_t)}\right]^{y_t}\left[1-\frac{1}{1+\exp(-\theta x_t)}\right]^{1-y_t}$$

where θ are the model parameters. The logistic function $p(y_t=1|x_t) = \frac{1}{1+\exp(-\theta x_t)}$ is conviniently bounded between 0 and 1.

We also assume a Gaussian prior

$$p(\theta) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{1}{2\sigma^2}(\theta-\mu)'(\theta-\mu)\right)$$

The goal of the analysis is then to compute the posterior distribution $p(\theta|x_{1:T},y_{1:T})$. This distribution will enable us to classify new data as follows

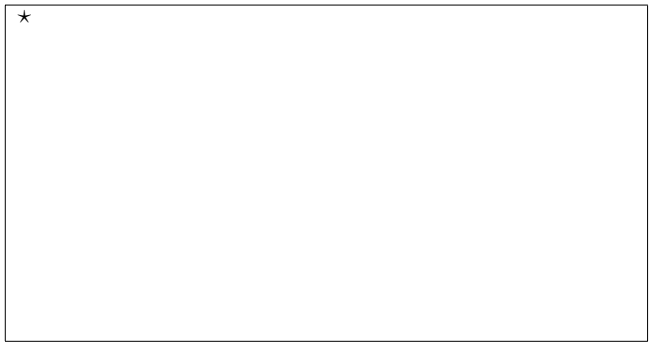
$$p(y_{T+1}|x_{1:T+1}) = \int_\Theta p(y_{T+1}|x_{T+1},\theta)p(\theta|x_{1:T},y_{1:T})d\theta$$

Bayes' rule gives us the following expression for the posterior

$$p(\theta|x_{1:T},y_{1:T}) \propto \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{1}{2\sigma^2}(\theta-\mu)'(\theta-\mu)\right) \times\prod_{t=1}^{T}\left[\frac{1}{1+\exp(-\theta' x)}\right]^{y_t}\left[1-\frac{1}{1+\exp(-\theta' x)}\right]^{1-y_t}$$

The problem is that in this case we can't solve the normalising integral analytically. So we have to use numerical methods — in this case importance sampling — to approximate $p(\theta|x_{1:T},y_{1:T})$. Note that we cannot sample from $p(\theta|x_{1:T},y_{1:T})$ directly because we don't know the normalising constant. So instead we sample from a proposal distribution $q(\theta)$ (say a Gaussian) and weight the samples using importance sampling. After obtaining N samples of θ from

the posterior, we can classify new data as follows

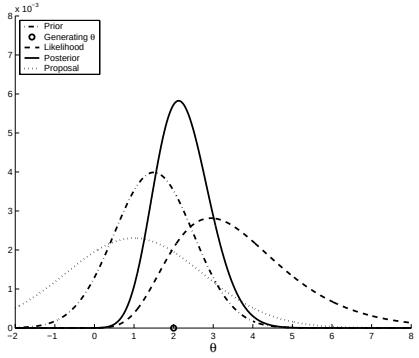


In your homework, you'll be given some data and the following prior and proposal:

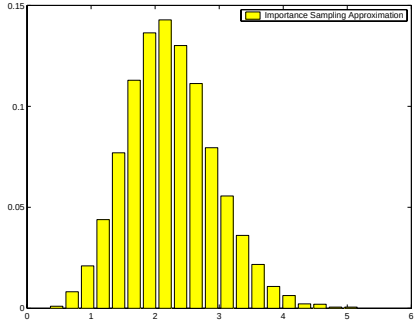
$$p(\theta) = \mathcal{N}(1, 1.5)$$

$$q(\theta) = \mathcal{N}(1, 3)$$

The prior, likelihood, posterior and proposal are shown in the following plot.



The empirical posterior (histogram approximation) obtained with importance sampling is:



3 MCMC Algorithms

MCMC is a strategy for generating samples $x^{(i)}$ while exploring the state space $\mathcal{X}$ using a Markov chain mechanism. This mechanism is constructed so that the chain spends more time in the most important regions. In particular, it is constructed so that the samples $x^{(i)}$ mimic samples drawn from the target distribution $p(x)$. (We reiterate that we use MCMC when we cannot draw samples from $p(x)$ directly, but can evaluate $p(x)$ up to a normalising constant.)

It is intuitive to introduce Markov chains on finite state spaces, where $x^{(i)}$ can only take s discrete values $x^{(i)} \in \mathcal{X} = \{x_1, x_2, \ldots, x_s\}$. The stochastic process $x^{(i)}$ is called a Markov chain if

$$p(x^{(i)}|x^{(i-1)}, \ldots, x^{(1)}) = T(x^{(i)}|x^{(i-1)}),$$

Google's PageRank is a good example of a Markov chain algorithm.

For any starting point, the chain will converge to the invariant distribution $p(x)$ (principal eigenvector), as long as T is a stochastic transition matrix that obeys the following properties:

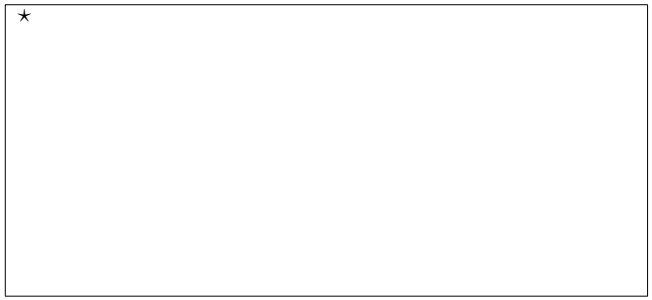
1. *Irreducibility*: For any state of the Markov chain, there is a positive probability of visiting all other states. That is, the matrix T cannot be reduced to separate smaller matrices, which is also the same as stating that the transition graph is connected.

2. *Aperiodicity*: The chain should not get trapped in cycles.

A sufficient, but not necessary, condition to ensure that a particular $p(x)$ is the desired invariant distribution is the following reversibility (detailed balance) condition

$$p(x^{(i)})T(x^{(i-1)}|x^{(i)}) = p(x^{(i-1)})T(x^{(i)}|x^{(i-1)}).$$

Summing both sides over $x^{(i-1)}$, gives us



MCMC samplers are irreducible and aperiodic Markov chains that have the target distribution as the invariant distribution. One way to design these samplers is to ensure that detailed balance is satisfied. However, it is also important to design samplers that converge quickly.

In continuous state spaces, the transition matrix T becomes an integral kernel K and $p(x)$ becomes the corresponding eigenfunction

$$\int p(x^{(i)})K(x^{(i+1)}|x^{(i)})dx^{(i)} = p(x^{(i+1)}).$$

The kernel K is the conditional density of $x^{(i+1)}$ given the value $x^{(i)}$. It is a mathematical representation of a Markov chain algorithm. In the following sections, we will see how to construct algorithmic versions of these kernels using a general recipe known as Metropolis-Hastings.

The Metropolis-Hastings Algorithm

The *Metropolis-Hastings* (MH) algorithm is the most popular class of MCMC methods. Most practical MCMC algorithms can be interpreted as special cases or extensions of this algorithm.

An MH step of invariant distribution $p(x)$ and proposal distribution $q(x^\star|\, x)$ involves sampling a candidate value $x^\star$ given the current value x according to $q(x^\star|\, x)$. The Markov chain then moves towards $x^\star$ with acceptance probability

$$\mathcal{A}(x, x^\star) = \min\{1, [p(x)q(x^\star|x)]^{-1}\, p(x^\star)q(x|x^\star)\}$$

Otherwise, it remains at x.

The pseudo-code is

1. Initialise $x^{(0)}$.
2. For $i = 0$ to $N - 1$
   - Sample $u \sim \mathcal{U}_{[0,1]}$.
   - Sample $x^\star \sim q(x^\star|x^{(i)})$.
   - If $u < \mathcal{A}(x^{(i)}, x^\star) = \min\left\{1, \frac{p(x^\star)q(x^{(i)}|x^\star)}{p(x^{(i)})q(x^\star|x^{(i)})}\right\}$

     $x^{(i+1)} = x^\star$

     else

     $x^{(i+1)} = x^{(i)}$

The following figure shows the results of running the MH algorithm with a Gaussian proposal distribution

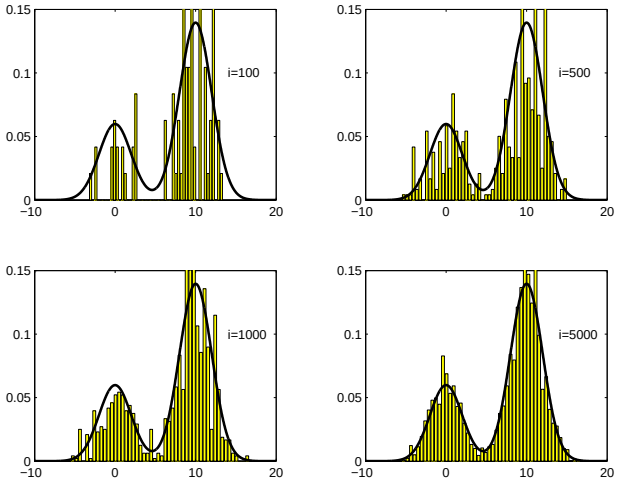
$$q(x^\star|x^{(i)}) = \mathcal{N}(x^{(i)}, 100)$$

and a bimodal target distribution

$$p(x) \propto 0.3\exp\left(-0.2x^2\right) + 0.7\exp\left(-0.2(x-10)^2\right)$$

for 5000 iterations. As expected, the histogram of the sam-

ples approximates the target distribution.



In the following class, we will analyse the MH algorithms and its variants, including the independence sampler, Metropolis Random Walk, and Gibbs sampler. We will also introduce particle filtering.

The transition kernel for the MH algorithm is

$$K_{\text{MH}}(x^{(i+1)}|x^{(i)}) = q(x^{(i+1)}|x^{(i)})\mathcal{A}(x^{(i)}, x^{(i+1)}) + \delta_{x^{(i)}}(x^{(i+1)})r(x^{(i)}),$$

where $r(x^{(i)})$ is the term associated with rejection

$$r(x^{(i)}) = 1 - \int_{\mathcal{X}-x^{(i)}} q(x^\star|x^{(i)})\mathcal{A}(x^{(i)}, x^\star)dx^\star.$$

⋆ Proof:

It is fairly easy to prove that the samples generated by MH algorithm will mimic samples drawn from the target distribution asymptotically. By construction, K_{MH} satisfies the detailed balance condition (prove this by substituting the expression for the MH kernel into the detailed balance equation)

$$p(x^{(i)})K_{\text{MH}}(x^{(i+1)}|x^{(i)}) \;=\; p(x^{(i+1)})K_{\text{MH}}(x^{(i)}|x^{(i+1)})$$

and, consequently, the MH algorithm admits $p(x)$ as invariant distribution. To show that the MH algorithm converges, we need to ensure that there are no cycles (aperiodicity) and that every state that has positive probability can be reached in a finite number of steps (irreducibility). Since the algorithm always allows for rejection, it follows that it is aperiodic. To ensure irreducibility, we simply need to make sure that the support of $q(\cdot)$ includes the support of $p(\cdot)$. Under these conditions, we obtain asymptotic convergence.

The *independent sampler* and the *Metropolis algorithm* are two simple instances of the MH algorithm. In the independent sampler the proposal is independent of the current state, $q(x^\star|x^{(i)}) = q(x^\star)$. Hence, the acceptance probability is

$$\mathcal{A}(x^{(i)}, x^\star) = \min\left\{1, \frac{p(x^\star)q(x^{(i)})}{p(x^{(i)})q(x^\star)}\right\} = \min\left\{1, \frac{w(x^\star)}{w(x^{(i)})}\right\}.$$

This algorithm is close to importance sampling, but now the samples are correlated since they result from comparing one sample to the other.

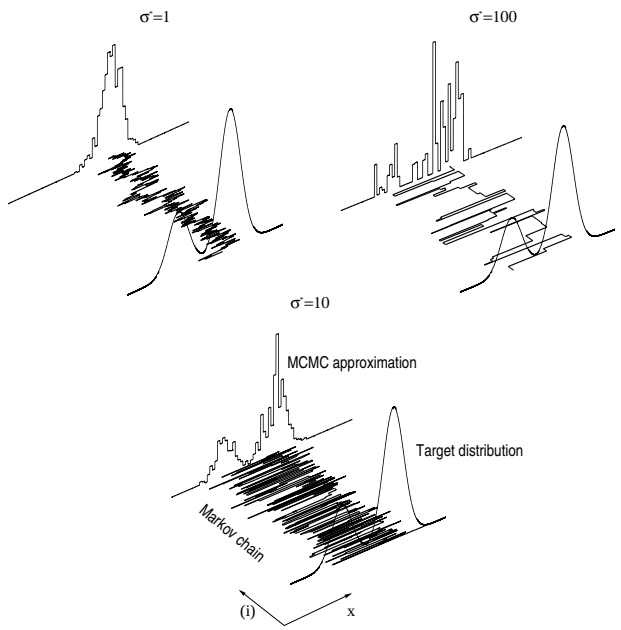
The Metropolis algorithm assumes a symmetric random walk proposal $q(x^\star|x^{(i)}) = q(x^{(i)}|x^\star)$ and, hence, the acceptance ratio simplifies to

$$\mathcal{A}(x^{(i)}, x^\star) = \min\left\{1, \frac{p(x^\star)}{p(x^{(i)})}\right\}.$$

Some properties of the MH algorithm are worth highlighting.

- Firstly, the normalising constant of the target distribution is not required. We only need to know the target distribution up to a constant of proportionality.
- Secondly, it is easy to simulate several independent chains in parallel.
- Lastly, the success or failure of the algorithm often hinges on the choice of proposal distribution.

Different choices of the proposal standard deviation $\sigma^\star$ lead to very different results. If the proposal is too narrow, only one mode of $p(x)$ might be visited. On the other hand, if it is too wide, the rejection rate can be very high, resulting in high correlations. If all the modes are visited while the acceptance probability is high, the chain is said to "mix" well. This is illustrated in the following figure



Simulated annealing for global optimization

Let us assume that instead of wanting to approximate $p(x)$, we want to find its global maximum. For example, if $p(x)$ is the likelihood or posterior distribution, we often want the ML and maximum *a posteriori* (MAP) estimates.

In simulated annealing, one runs a non-homogeneous Markov chain whose invariant distribution at iteration i is no longer equal to $p(x)$, but to

$$p_i(x) \propto p^{1/T_i}(x),$$

where T_i is a decreasing cooling schedule with $\lim_{i\to\infty} T_i = 0$. The reason for doing this is that, under weak regularity assumptions on $p(x)$, $p^\infty(x)$ is a probability density that concentrates itself on the set of global maxima of $p(x)$. The simulated annealing involves, therefore, just a minor modification of standard MCMC algorithms as shown by the following pseudo-code.

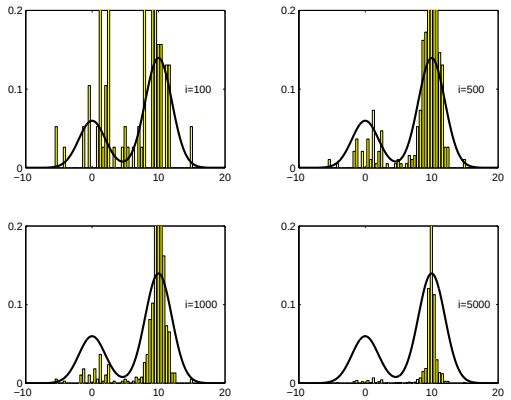
1. Initialise $x^{(0)}$ and set $T_0 = 1$.
2. For $i = 0$ to $N - 1$
   - Sample $u \sim \mathcal{U}_{[0,1]}$.
   - Sample $x^\star \sim q(x^\star|x^{(i)})$.
   - If $u < \mathcal{A}(x^{(i)}, x^\star) = \min\left\{1, \frac{p^{\frac{1}{T_i}}(x^\star)q(x^{(i)}|x^\star)}{p^{\frac{1}{T_i}}(x^{(i)})q(x^\star|x^{(i)})}\right\}$

     $$x^{(i+1)} = x^\star$$

     else

     $$x^{(i+1)} = x^{(i)}$$

   - Set T_{i+1} according to a chosen cooling schedule.



The Gibbs sampler

Suppose we have an n-dimensional vector x and the expressions for the full conditionals $p(x_j|x_1,\dots,x_{j-1},x_{j+1},\dots,x_n)$. In this case, it is often advantageous to use the following proposal distribution for $j = 1,\dots,n$

$$q(x^\star|x^{(i)}) = \begin{cases} p(x_j^\star|x_{-j}^{(i)}) & \text{If } x_{-j}^\star = x_{-j}^{(i)} \\ 0 & \text{Otherwise.} \end{cases}$$

The corresponding acceptance probability is:

⋆ Proof:

That is, the acceptance probability for each proposal is one and, hence, the deterministic scan Gibbs sampler algorithm is often presented as shown below:

1. Initialise $x_{0,1:n}$.
2. For $i = 0$ to $N-1$
   - Sample $x_1^{(i+1)} \sim p(x_1|x_2^{(i)},x_3^{(i)},\dots,x_n^{(i)})$.
   - Sample $x_2^{(i+1)} \sim p(x_2|x_1^{(i+1)},x_3^{(i)},\dots,x_n^{(i)})$.
   - $\vdots$
   - Sample $x_j^{(i+1)} \sim p(x_j|x_1^{(i+1)},\dots,x_{j-1}^{(i+1)},x_{j+1}^{(i)},\dots,x_n^{(i)})$.
   - $\vdots$
   - Sample $x_n^{(i+1)} \sim p(x_n|x_1^{(i+1)},x_2^{(i+1)},\dots x_{n-1}^{(i+1)})$.

Since the Gibbs sampler can be viewed as a special case of the MH algorithm, it is possible to introduce MH steps into the Gibbs sampler. It is also possible to group variables in blocks an update them simultaneously.

Lecture 10 - *Neural Networks and Back-Propagation*

OBJECTIVE: In this lecture, we discuss gradient based optimization for learning nonlinear classification and regression models.

◇ GRADIENT DESCENT TECHNIQUES

Searching for a good solution can be interpreted as looking for a minimum of some error (loss) function in parameter space.

⋆

The *gradient* is the vector of derivatives:

$$\nabla E = \left(\frac{dE}{d\theta_1} \quad \dots \quad \frac{dE}{d\theta_d} \right)$$

The gradient vector is orthogonal to the contours. Hence, to minimise the error, we follow the gradient (the direction of maximum decrease in error).

Let's go back to the linear model $Y = X\theta$ with quadratic error function $E = (Y - X\theta)'(Y - X\theta)$. The gradient for this model is:

$\star$

$$\nabla E =$$

The gradient descent learning rule, at iteration t, is:

$$\begin{aligned}\theta^{(t)} &= \theta^{(t-1)} + \alpha \nabla E \\ &= \theta^{(t-1)} + \alpha X'(Y - X\theta^{(t-1)})\end{aligned}$$

where α is a user-specified learning rate. See the textbook of Jordan for a convergence proof of this rule. The proof shows that, under certain regularities on α, $\theta^{(\infty)} = (X'X)^{-1}X'Y$. Not surprising!

In some situations, we might want to learn the parameters by going over the data **on-line**:

$$\theta^{(t)} = \theta^{(t-1)} + \alpha x^{(t)}(y^{(t)} - x^{(t)}\theta^{(t-1)})$$

This is the **least mean squares** algorithm. This learning rule is a **stochastic approximation** technique also known as the **Robbins-Monro** procedure. It's stochastic because the data is assumed to come from a stochastic process.

If α decreases with rate $1/n$, one can show that this algorithm converges. If the θ vary "slowly" with time, it is also possible to obtain convergence proofs. Later, we will see that this algorithm can be exploited to obtain efficient on-line EM algorithms and adaptive sampling schemes.

The **Newton-Raphson** algorithm uses the gradient learning rule, with the inverse Hessian matrix in place of α:

$$\theta^{(t)} = \theta^{(t-1)} + H^{-1} \nabla E$$
$$H = \frac{\partial^2 E}{\partial \theta^2}$$

⋆ For the linear model we have:

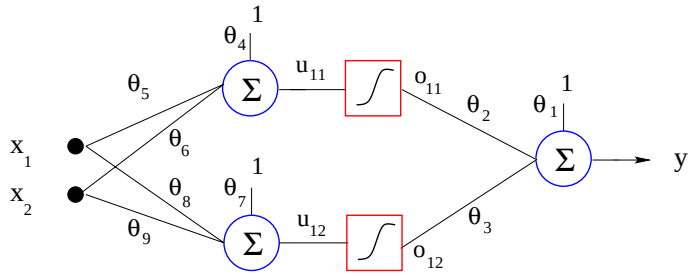
$$H =$$

$$\theta^{(t)} = \theta^{(t-1)} + H^{-1} \nabla E$$

Note that α is a scalar, while H is a large matrix. So there is a trade-off between speed of convergence and storage.

◇ ARTIFICIAL NEURAL NETWORKS

Gradient descent techniques allow us to learn complex, nonlinear models. Assume we are given the data $\{x_{1:n}, y_{1:n}\}$ and want to come up with a nonlinear mapping $\widehat{y} = f(x, \theta)$, where θ is obtained my minimising a loss function, say quadratic $E = (y - f(x, \theta))'(y - f(x, \theta))$. Our mapping will be the **artificial neural network (multi-layer perceptron)** depicted below:



It can be used to carry out non-linear regression (output neuron is linear) and nonlinear classification (output neuron

is sigmoidal). Mathematically, this network is equivalent to:

$$\widehat{y} = \phi_j\left(\phi_i\left(x\theta_j\right)\theta_i\right)$$

where $\phi(\cdot)$ is the sigmoidal (logistic) function:

$$\phi_i\left(X\theta_j\right) = \frac{1}{1+e^{-X\theta_j}}$$

⋆

The **synaptic weights** θ can be learned by following gradients:

$$\theta^{(t+1)} = \theta^{(t)} + \alpha(y-\widehat{y})^2\frac{\partial\widehat{y}}{\partial\theta^{(t)}}$$

where $\widehat{y} = f(x,\theta^{(t)})$. We proceed to compute the derivatives.

The **output layer** mapping is given by:

$$\widehat{y} = \theta_1 + \theta_2 o_{11} + \theta_3 o_{12}$$

and consequently, the derivatives with respect to the weights are given by:

⋆

$$\frac{\partial\widehat{y}}{\partial\theta_1} =$$
$$\frac{\partial\widehat{y}}{\partial\theta_2} =$$
$$\frac{\partial\widehat{y}}{\partial\theta_3} =$$

The **hidden layer** mapping for the top neuron is:

$$o_{11} = \frac{1}{1+\exp(-u_{11})} \qquad \text{where } u_{11} = \theta_4 + \theta_5 x_1 + \theta_6 x_2$$

Note that

$$\frac{\partial o_{11}}{\partial u_{11}} = o_{11}(1-o_{11})$$

⋆ The derivatives with respect to the weights are:

$$\frac{\partial \widehat{y}}{\partial \theta_4} = \frac{\partial \widehat{y}}{\partial o_{11}} \frac{\partial o_{11}}{\partial u_{11}} \frac{\partial u_{11}}{\partial \theta_4}$$

$$= $$

$$\frac{\partial \widehat{y}}{\partial \theta_5} = $$

$$\frac{\partial \widehat{y}}{\partial \theta_6} = $$

The derivatives with respect to the weights of the other hidden layer neuron can be calculated following the same procedure, known as **back-propagation**. Once we have all the derivatives, we can use either steepest descent or Newton-Raphson to update the weights.