In [3]:

```
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:80% !important; }</style>"))
```

THIS MATERIAL IS UNDER DEVELOPMENT AND IS NOT YET IN A READABLE STATE. IF YOU COPY, DISTRIBUTE OR THIS MATERIAL, I WOULD APPRECIATE IF YOU ADD A CITATION.

Best regards, Jonne Pohjankukka.

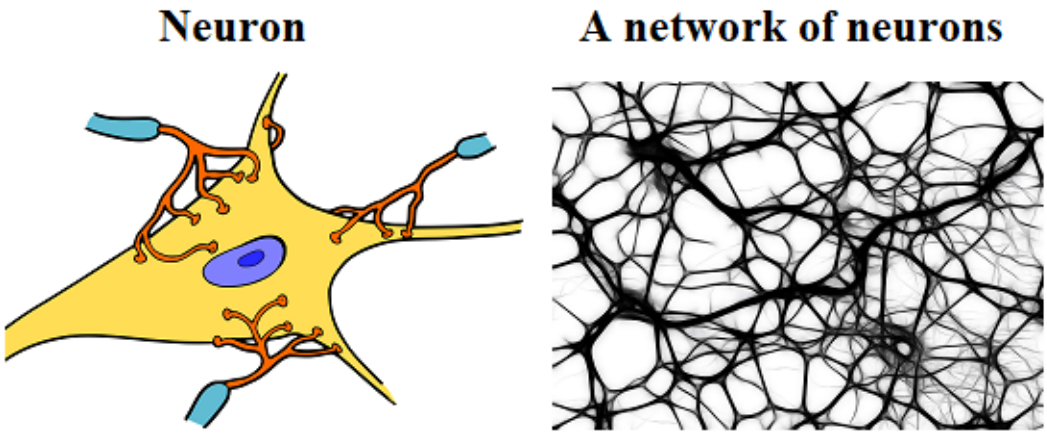
0. Introduction

The purpose of this tutorial is to given an introductory on the subject of Hamiltonian Monte Carlo and deep neural networks. We will begin this tutorial by going through the basics concepts from mechanics and propability theory, which are required for understanding the subject. I must say, the theory behind especially the Hamiltonian Monte Carlo method is massive. The subject covers disciplins ranging from the study of random processes (Brownian motion) to investigation of systems of particles in physics (statistical mechanics, quantum physics). So there's a lot to chew on with this subject, but the reward will be very satisfying.

In a summary, this tutorial brings together subjects from mathematical optimization, study of stochastic processes, Brownian/Wiener processes, Markov chains, statistical simulation inference, Bayesian modeling, statistical mechanics/thermodynamics, concept of entropy, calculus of variations, neural systems and hierarchical convolution-based learning.

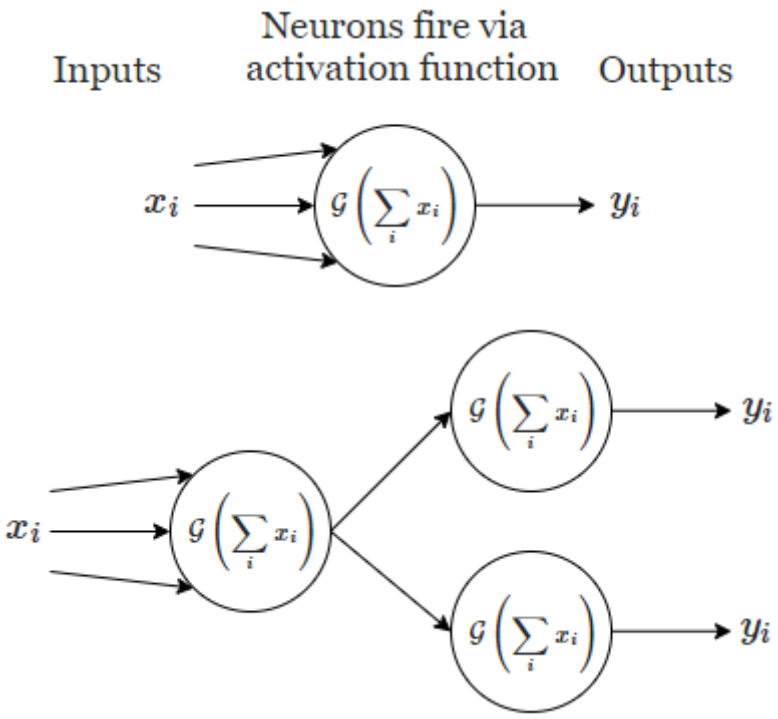
1. Neural networks

Basic information, biological inspiration



1.1 Perceptron

Perceptron function



1.2 From single perceptron to multiple neurons

A group of perceptrons --> network

2. Convolutional neural networks

Now that we have a good conceptual understanding of a basic neural network (or multilayer perceptron), it is time to turn our attention to a specific type of a neural network which over the last few years has been dominating image-based pattern recognition. I'm talking about of course, the convolutional neural networks (CNNs). Whereas the basic neural network learns a complicated function from the data, a CNN learns filters from the data and then utilizes these learned filters in pattern recognition. Of course in the end both basic neural network and CNN learn the same thing, a function, so there is no real big difference between the two, just the way on how the neural network is organized and structured.

But before we can get into the meat of the matter, we must first understand what is convolution? After all it's a pretty important thing since it's in the name of the CNN ^^

2.1 Convolution

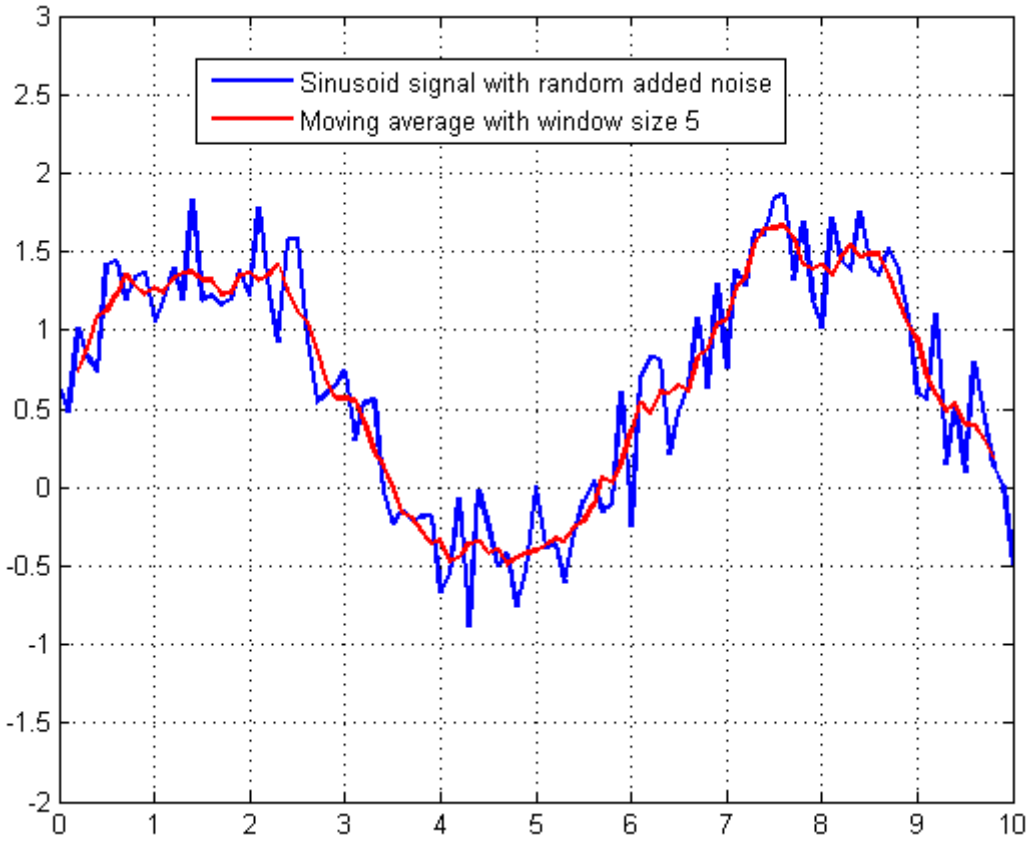
So what is a convolution? It turns out that convolution is an extremely simple operation, even though you might initially think otherwise when looking e.g. into wikipedia article (https://en.wikipedia.org/wiki/Convolution) on the subject. In words, convolution is about having two functions f and g, multiplying them elementwise and then summing up all the multiplications, that's all it is! You can think of convolution as a filter, which takes two signals (functions) as inputs and outputs a new function. This new function is the result of filtering the function f with function g. In mathematical terms, the definitions for convolution is:

$$(f * g)(y) = \int_{-\infty}^{\infty} f(x)\, g(y - x)\, dx$$

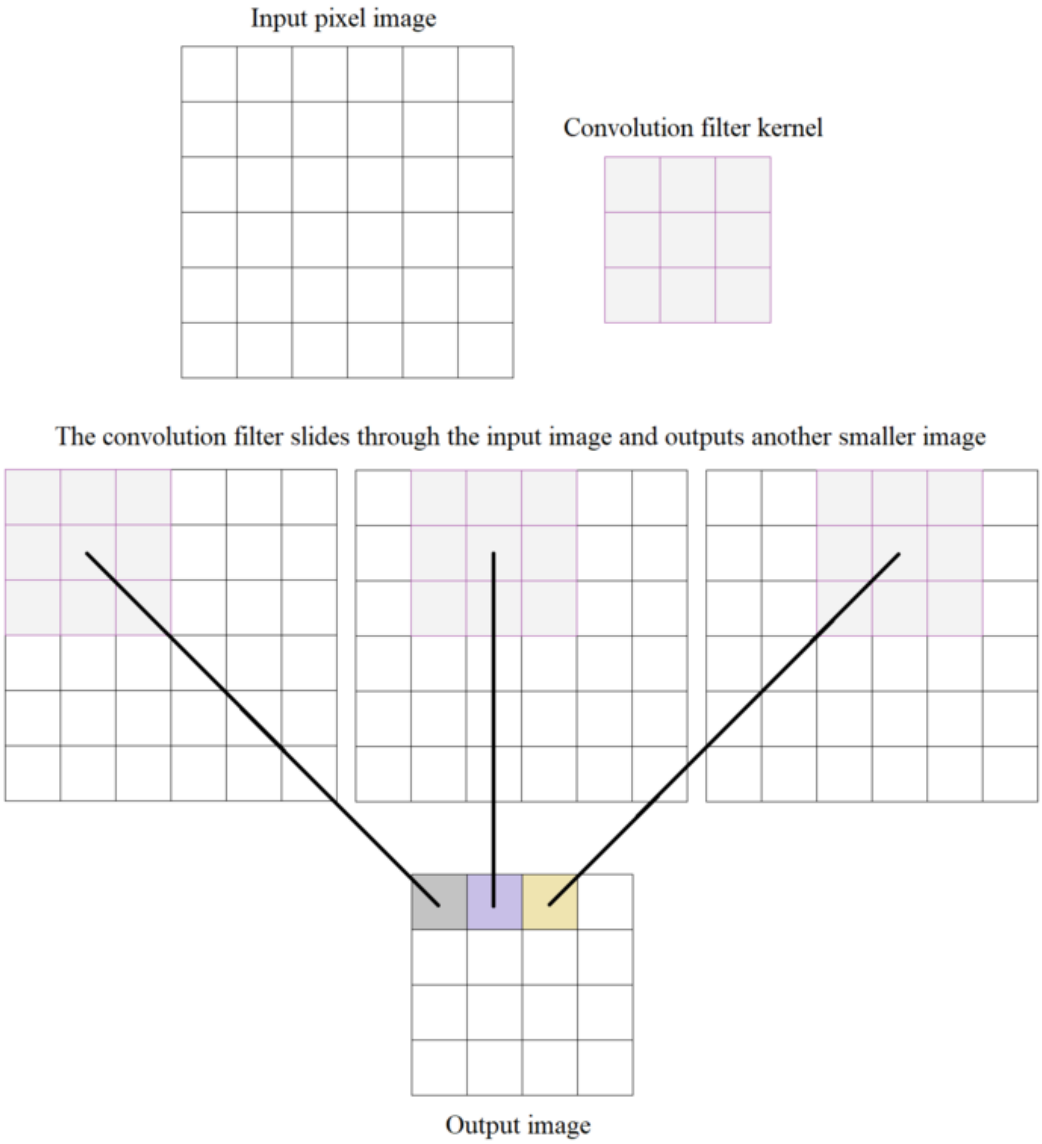
and the multidimensional analogue is:

$$(f * g)(\mathbf{y}) = \int_{\mathbb{R}^d} f(\mathbf{x})\, g(\mathbf{y} - \mathbf{x})\, d\mathbf{x}.$$

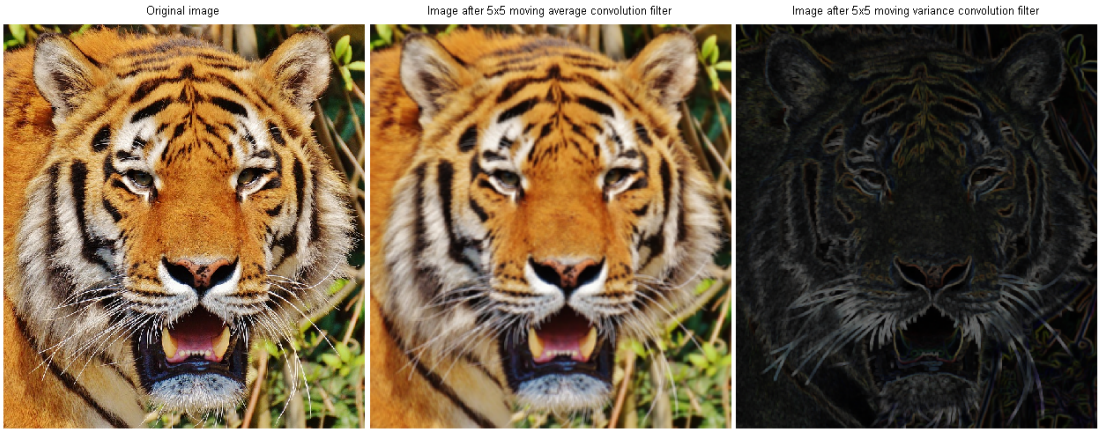
The asterisk sign $*$ denotes the convolution operation. The new function resulting from the convolution is $(f * g)(y)$. I think one of the easiest examples of convolution is [the moving average (https://en.wikipedia.org/wiki/Moving_average)](). As its name says, the moving average simply produces a smoothed version of a given signal. For example, in the below figure I have applied the moving average convolution to a sine function with added random noise:



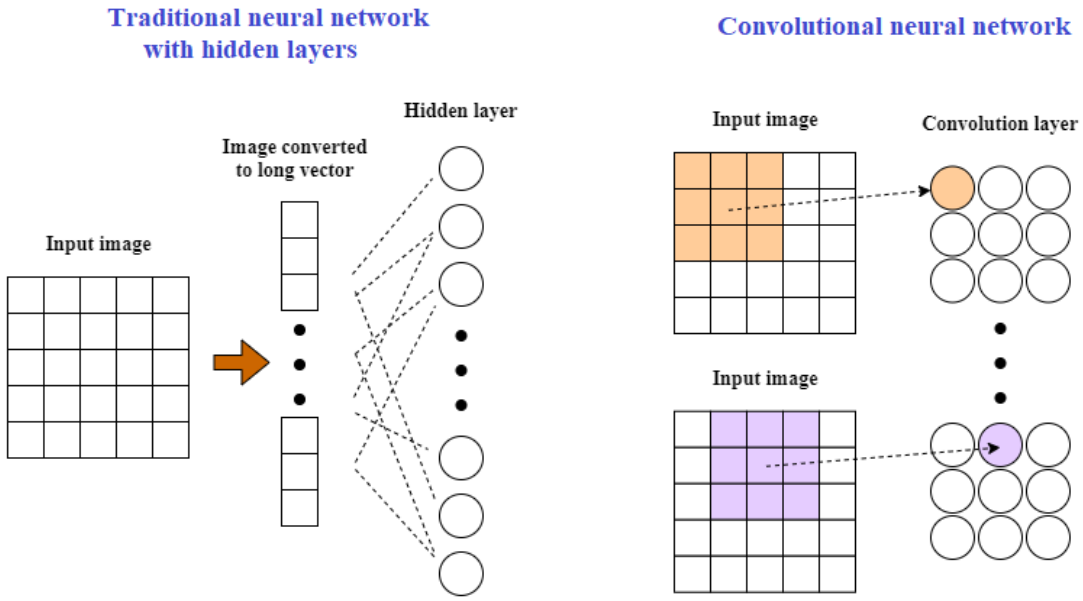
In other words, the sine function here is the input signal f and the moving average filter g is what smoothes the sine function, producing the red curve (output function) in the figure. The above example was about 1-dimensional convolution but you can generalize this to any dimension. The CNNs apply (surprise surprise) 2-dimensional convolution since the data sets consist usually from images. The 2-dimensional convolution is exactly the same as 1D, but instead on summing and multiplying in one dimension you do the same thing in two dimensions. I think the next figure explains this the best:



In the next figure of a tiger, I have applied two different kinds of 2-dimensional convolution filters:



2.2 Grouping neurons into convolution filters



3. Training neural networks: back propagation

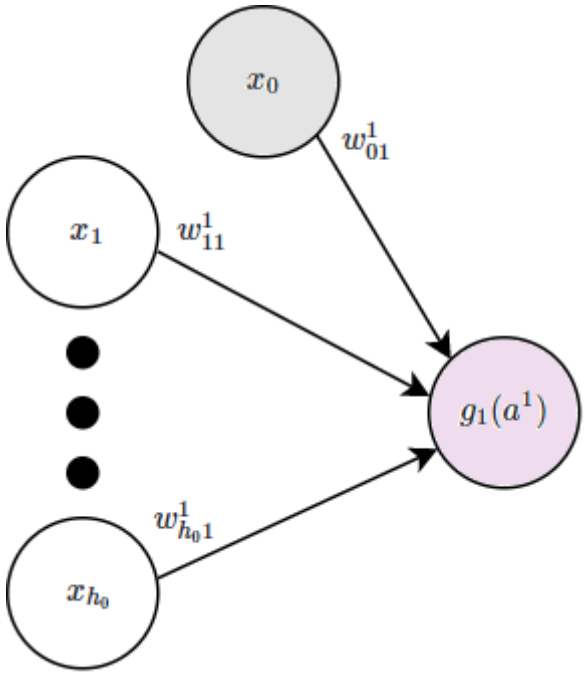
3.1 gradient descent

3.2 stochastic gradient descent

3.3 back propagation algorithm: derivation

3.4 Example: training multilayer perceptron to learn a sine function with C++

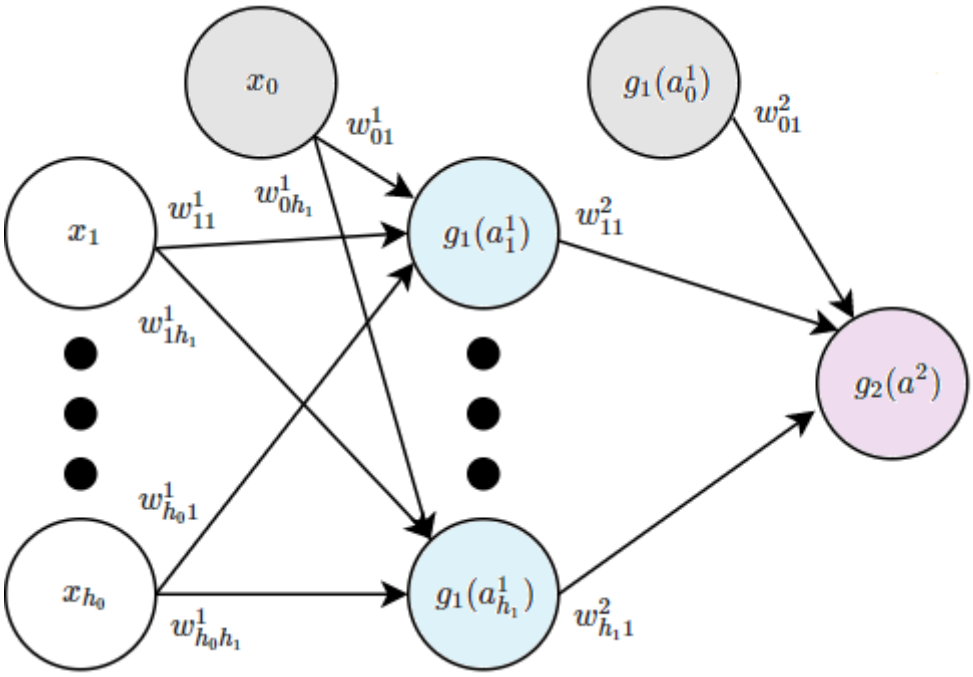
3.5 back propagation for CNN



$h_0 - 1$

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}\left\{y_i - g_1\left(a_i^1\right)\right\}^2 = \frac{1}{2}\sum_{i=1}^{N}\left\{y_i - g_1\left(\sum_{j=0}^{h_0} w_{j1}^1 x_{ij}\right)\right\}^2$$

$$\frac{\partial E}{\partial w_{k1}^1} = -\sum_{i=1}^{N}\left\{y_i - g_1\left(a_i^1\right)\right\}\frac{\partial g_1}{\partial a_i^1}\frac{\partial a_i^1}{\partial w_{k1}^1} = -\sum_{i=1}^{N}\left\{y_i - g_1\left(a_i^1\right)\right\}\frac{\partial g_1}{\partial a_i^1}x_{ik}$$

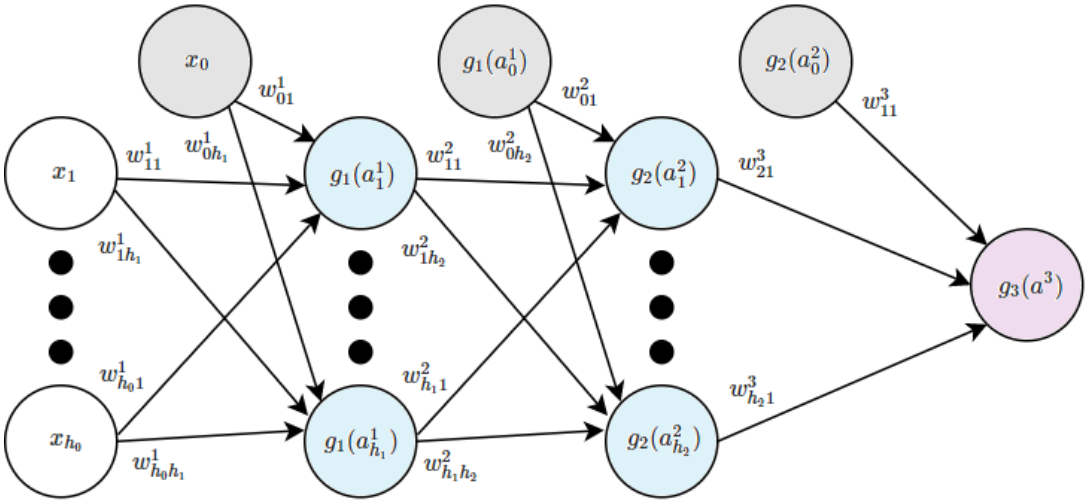


$h_0 - h_1 - 1$

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}\left\{y_i - g_2\left(a_i^2\right)\right\}^2 = \frac{1}{2}\sum_{i=1}^{N}\left\{y_i - g_2\left(\sum_{j=0}^{h_1} w_{j1}^2 g_1(a_j^1)\right)\right\}^2 = \frac{1}{2}\sum_{i=1}^{N}$$

$$\frac{\partial E}{\partial w_{k1}^2} = -\sum_{i=1}^{N}\left\{y_i - g_2\left(a_i^2\right)\right\}\frac{\partial g_2}{\partial a_i^2}\frac{\partial a_i^2}{\partial w_{k1}^2} = -\sum_{i=1}^{N}\left\{y_i - g_2\left(a_i^2\right)\right\}\frac{\partial g_2}{\partial a_i^2}g_1(a_k^1)$$

$$\frac{\partial E}{\partial w_{kp}^1} = -w_{p1}^2\sum_{i=1}^{N}\left\{y_i - g_2\left(a_i^2\right)\right\}\frac{\partial g_2}{\partial a_i^2}\frac{\partial g_1}{\partial a_p^1}\frac{\partial a_p^1}{\partial w_{kp}^1} = -w_{p1}^2\sum_{i=1}^{N}\left\{y_i - g_2\left(a_i^2\right)\right\}\frac{\partial g_2}{\partial a_i^2}$$



$h_0 - h_1 - h_2 - 1$

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}\left\{y_i - g_3\left(a_i^3\right)\right\}^2 = \frac{1}{2}\sum_{i=1}^{N}\left\{y_i - g_3\left(\sum_{j=0}^{h_2} w_{j1}^3$$

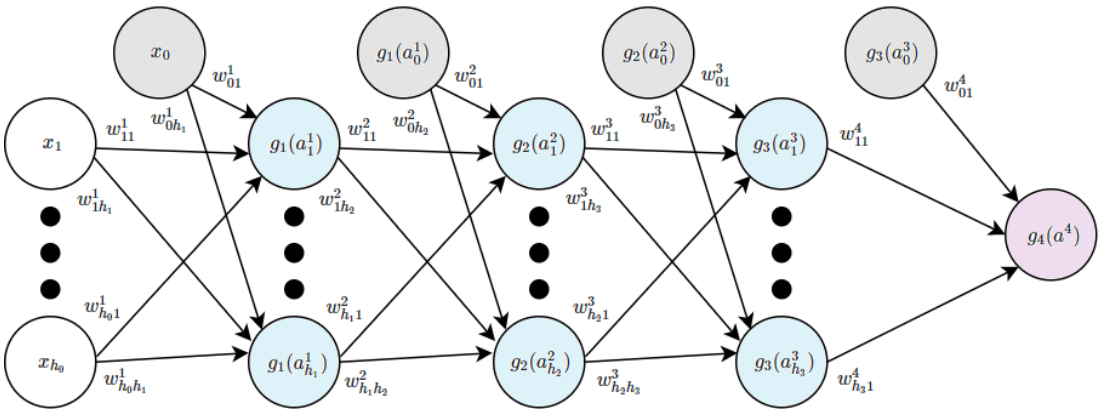
$$\frac{1}{2}\sum_{i=1}^{N}\left\{y_i - g_3\left(\sum_{j=0}^{h_2} w_{j1}^3 g_2\left(\sum_{l=0}^{h_1} w_{lj}^2 g_1\left(a_l^1\right)\right)\right)\right\}^2 = \frac{1}{2}\sum_{i=1}^{N}\left\{y_i - g_3\left(\sum_{j=0}^{h_2} w_{j1}^3\right.\right.$$

$$\frac{\partial E}{\partial w_{k1}^3} = -\sum_{i=1}^{N}\left\{y_i - g_3\left(a_i^3\right)\right\}\frac{\partial g_3}{\partial a_i^3}\frac{\partial a_i^3}{\partial w_{k1}^3} = -\sum_{i=1}^{N}\left\{y_i - g_3\left(a_i^3\right)\right\}\frac{\partial g_3}{\partial a_i^3}g_2(a_k^2)$$

$$\frac{\partial E}{\partial w_{kp}^2} = -w_{p1}^3\sum_{i=1}^{N}\left\{y_i - g_3\left(a_i^3\right)\right\}\frac{\partial g_3}{\partial a_i^3}\frac{\partial g_2}{\partial a_p^2}\frac{\partial a_p^2}{\partial w_{kp}^2} = -w_{p1}^3\sum_{i=1}^{N}\left\{y_i - g_3\left(a_i^3\right)\right\}\frac{\partial g_3}{\partial a_i^3}$$

$$\frac{\partial E}{\partial w_{kp}^1} = -\sum_{i=1}^{N}\left\{y_i - g_3\left(a_i^3\right)\right\}\frac{\partial g_3}{\partial a_i^3}\left[\sum_{j=0}^{h_2} w_{j1}^3\frac{\partial g_2}{\partial a_j^2}w_{pj}^2\frac{\partial g_1}{\partial a_p^1}\frac{\partial a_p^1}{\partial w_{kp}^1}\right]$$

$$= -\sum_{i=1}^{N}\left\{y_i - g_3\left(a_i^3\right)\right\}\frac{\partial g_3}{\partial a_i^3}\left[\sum_{j=0}^{h_2} w_{j1}^3\frac{\partial g_2}{\partial a_j^2}w_{pj}^2\frac{\partial g_1}{\partial a_p^1}x_{ik}\right]$$



$h_0 - h_1 - h_2 - h_3 - 1$

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}\left\{y_i - g_4\left(a_i^4\right)\right\}^2 = \frac{1}{2}\sum_{i=1}^{N}\left\{ y_i - g_4\left(\sum_{j=0}^{h_3} w_j^4$$

$$\frac{1}{2}\sum_{i=1}^{N}\left\{ y_i - g_4\left(\sum_{j=0}^{h_3} w_{j1}^4 g_3\left(\sum_{l=0}^{h_2} w_{lj}^3 g_2\left(a_l^2\right)\right)\right)\right\}^2 = \frac{1}{2}\sum_{i=1}^{N}\left\{ y_i - g_4\left(\sum_{j=0}^{h_3} w_{j1}^4,\right.\right.$$

$$= \frac{1}{2}\sum_{i=1}^{N}\left\{ y_i - g_4\left(\sum_{j=0}^{h_3} w_{j1}^4 g_3\left(\sum_{l=0}^{h_2} w_{lj}^3 g_2\left(\sum_{d=0}^{h_1} w_{dl}^2 g_1\left(\sum_{s=0}^{h_0} w_{sd}^1 x_{is}\right)\right)\right)\right)\right\}^2$$

$$\frac{\partial E}{\partial w_{k1}^4} = -\sum_{i=1}^{N}\left\{y_i - g_4\left(a_i^4\right)\right\}\frac{\partial g_4}{\partial a_i^4}\frac{\partial a_i^4}{\partial w_{k1}^4} = -\sum_{i=1}^{N}\left\{y_i - g_4\left(a_i^4\right)\right\}\frac{\partial g_4}{\partial a_i^4} g_3(a_k^3)$$

$$\frac{\partial E}{\partial w_{kp}^3} = -w_{p1}^4\sum_{i=1}^{N}\left\{y_i - g_4\left(a_i^4\right)\right\}\frac{\partial g_4}{\partial a_i^4}\frac{\partial g_3}{\partial a_p^3}\frac{\partial a_p^3}{\partial w_{kp}^3} = -w_{p1}^4\sum_{i=1}^{N}\left\{y_i - g_4\left(a_i^4\right)\right\}\frac{\partial g_4}{\partial a_i^4}$$

$$\frac{\partial E}{\partial w_{kp}^2} = -\sum_{i=1}^{N}\left\{y_i - g_4\left(a_i^4\right)\right\}\frac{\partial g_4}{\partial a_i^4}\left[\sum_{j=0}^{h_3} w_{j1}^4\frac{\partial g_3}{\partial a_j^3} w_{pj}^3\frac{\partial g_2}{\partial a_p^2}\frac{\partial a_p^2}{\partial w_{kp}^2}\right]$$

$$= -\sum_{i=1}^{N}\left\{y_i - g_4\left(a_i^4\right)\right\}\frac{\partial g_4}{\partial a_i^4}\left[\sum_{j=0}^{h_3} w_{j1}^4\frac{\partial g_3}{\partial a_j^3} w_{pj}^3\frac{\partial g_2}{\partial a_p^2} x_{ik}\right]$$

$$\frac{\partial E}{\partial w_{kp}^1} = -\sum_{i=1}^{N}\left\{y_i - g_4\left(a_i^4\right)\right\}\frac{\partial g_4}{\partial a_i^4}\left[\sum_{j=0}^{h_3} w_{j1}^4\frac{\partial g_3}{\partial a_j^3}\left(\sum_{l=0}^{h_2} w_{lj}^3\frac{\partial g_2}{\partial a_l^2} w_{pl}^2\frac{\partial g_1}{\partial a_p^1} x_{ik}\right)\right]$$

4. MECHANICS

4.1 Newtonian / classical mechanics

In classical or Newtonian mechanics, the elementary relation between an object and an exerted force is describoed by the Newton's 2nd law:

$$F = ma,$$

which relates the force F exerted on an object with mass m and acceleration a. Force is measured in the units of Newton's (N) and 1 Newton can be interpreted in the following way: "One Newton, is the force required for causing an acceleration of $1\frac{m}{s^2}$ on an object with mass of 1 kg". To give an example, say we have an object with mass 10 kg and we wish to cause acceleration of $5\frac{m}{s^2}$ on the object. We thus need to apply the force:

$$F = 10 \cdot 5\ \frac{kg \cdot m}{s^2} = 50N,$$

on the object. So far, we have been talking about a single object or a particle, but what about when we have a system of particles or objects? How can we model the dynamics of a system of particles? Well, we simply sum up all the forces exerted on the system (by system I mean all the particles) and so the dynamics of group of particles is defined by the equation:

$$\sum_{i=1}^{n} F_i = \sum_{i=1}^{n} m_i a_i.$$

This equation includes both the internal and external forces on the system and constraint forces. Our task now is to use this equation and explain the dynamics of the system. One can quickly agree however, that this is a rather difficult problem: we have a multitude of particles in our hand with many different forces acting on the system. It is not thus easy to explain the dynamics of such a system using Newtonian mechanics. This is where we bring in the Lagrangian and Hamiltonian mechanics, which are equivalent to the Newtonian mechanics in describing the dynamics of the system, but they offer a much easier way to do this. Plus, they give us more information on the dynamics of the system than Newtonian mechanics does.

Example of Newtonian mechanics: Atwood machine

To give an example of Newtonian mechanics, we will consider the classical example of the Atwood machine (1784, by George Atwood). The Atwoow machine is a laboratory experiment for verifying the mechanical laws of motion in a constant acceleration case.

In a basic Atwood machine example, we consider two objects with masses m_1 kg and m_2 kg, connected by an inextensible massless string over an ideal massless pulley. In this example, we wish to investigate the acceleration a of the 'object-string-object'-system.

First of all, lets find the forces acting on the system. We assume for the sake of simplicity, that the string produces a consant force of T to the two objects. Also, we assume that the acceleration of the system is positive $(a > 0)$ when object with mass m_1 is falling down and negative $(a < 0)$ if it is rising up. In addition to the string force T, the forces acting on the objects are their weights due to gravity, that is:

$$W_1 = m_1 g \qquad W_2 = m_2 g,$$

where $g = 9.8 \ \frac{m}{s^2}$ is the acceleration due to gravity. So the total forces acting on the two objects are:

$$W_1 - T = m_1 g - T = m_1 a \qquad T - W_2 = T - m_2 g = m_2 a,$$

and if we add these two equations we get:

$$m_1 g - T + T - m_2 g = m_1 a + m_2 a,$$

from which we easily get that the acceleration of the 'object-string-object'-system is:

$$a = g \frac{m_1 - m_2}{m_1 + m_2}.$$

To make this more concrete with real numbers, let $m_1 = 1.1$ kg and $m_2 = 1$ kg. We thus get that the acceleration of the system is:

$$a = 9.8 \ \frac{m}{s^2} \frac{(1.1 - 1)kg}{(1.1 + 1)kg} \approx 0.048 \times 9.8 \frac{m}{s^2} \approx 0.47 \frac{m}{s^2}.$$

To get a better feeling of Atwood machine, please refer to the simulation by Andrew Duffy: Atwood machine (http://physics.bu.edu/~duffy/HTML5/Atwoods_machine.html).

4.2 Lagrangian mechanics

In mechanics, we are interested in the motion of objects: how fast a car drives, how the earth orbits the sun, the oscillating motion of a pendulum, etc. From a simple applied point of view, Lagrangian mechanics is just a different way to approach a given mechanical problem. Let us take as an example the motion of a pendulum. Our goal is to describe how the pendulum will move.

In Newtonian mechanics (the “normal” mechanics taught in high school), we would start by drawing a diagram with multiple arrows for all the forces which are acting on the pendulum. We can then find how the pendulum is moving by using Newton’s second law: F=ma. More generally in Newtonian mechanics, we take Newton’s three laws as fundamental laws of nature and try to derive everything else from there. It is centered around forces, since these are ultimately used to figure out the trajectories.

In Lagrangian mechanics, things work differently. To obtain the same result, we start by calculating the kinetic and potential energy of the pendulum. Instead of Newton’s three laws, we assume that there is another fundamental law of nature: the principle of least action. According to this principle, we can calculate the motion by minimizing a certain quantity, called the action, that is related to the two forms of energy mentioned above. Lagrangian mechanics therefore is centered around energies. Forces are no longer needed to determine the motion of objects.

Both variants of course lead to the same trajectory for the pendulum. It can be shown more generally that these two formalism are equivalent. However, they each are better suited for certain types of problems.

Besides being more convenient to solve some problems, there is a much deeper reason for why it is a good idea to introduce Lagrangian mechanics. It turns out that many of the fundamental laws of physics can be described by such a principle of least action. To give you a taste, here is the so called “Lagrangian” of the standard model (the action we try to minimize in the principle of least action is the integral of this beast):

As in Newtonian mechanics, we begin the Lagrangian formulation from the equation (with vector quantities):

$$\sum_{i=1}^{n} F_i = \sum_{i=1}^{n} m_i a_i,$$

which describes the dynamics of a system of n particles. Furthermore, we decompose the forces into two components, the applied forces $F_i^{(a)}$ and forces due to constraints f_i, and so we get:

$$\sum_{i=1}^{n} \left(F_i^{(a)} + f_i \right) = \sum_{i=1}^{n} m_i a_i = \sum_{i=1}^{n} \dot{p}_i,$$

where we have included the moment $p_i = m_i v_i$ notation, where v_i is the velocity of particle i. That is,

$$\dot{p}_i = \frac{dp_i}{dt} = \frac{d(m_i v_i)}{dt} = m_i \frac{dv_i}{dt} = m_i a_i.$$

Thus we have for the equation of motion of the system that:

$$\sum_{i=1}^{n} \left(F_i^{(a)} + f_i - \dot{p}_i \right) = \mathbf{0}.$$

An example of $F_i^{(a)}$ could be caused e.g.via a push by an external agent to the system. An example of a force of constraint f_i could be e.g. gravity which forces particle i to stay on a plane. Another easier example is perhaps to think of a rigid-body movement. The whole system experiences a force by an external agent and thus each particle i experiences a force, and each particle of the system is constrained to its own relative position wit respect to other particles of the system.

Next, we introduce more factors into the game. The symbols $r_1, r_2, \ldots, r_n$ refer to the position vectros of the n particles and time is denoted as t. The constraints imposed on the system on n particles are represented by the set of equations:

$$f(r_1, r_2, r_3, \ldots, t) = 0,$$

which are called holonomic constraints. If we have k of these holonomic constraint equations, then we can use them to eliminate k of the $3n$ (each particle has three coordinates) position coordinates of the particle system and we are left with $3n - k$ coordinates which we can select **independently** with respect to each other. These remaining $3n - k$ coordinates (denoted as q_i) are called generalized coordinates and they implicitly contain the constraints of the system, which is defined by the k holonimic equations. That is, we now have:

$$\begin{aligned} r_1 &= r_1(q_1, q_2, \ldots, q_{3n-k}, t) \\ &\vdots \\ r_n &= r_n(q_1, q_2, \ldots, q_{3n-k}, t). \end{aligned}$$

Next we introduce the concept of virtual displacement, which refers to the a change in the configuration of the system as the result of any arbitrary infinitesimal change of the coordinates δr_i, consistent with the forces and constraints imposed on the system at the given time instant t. By now taking the dot product of above equation with δr_i in each particle, we get that:

$$\sum_{i=1}^{n} \left(F_i^{(a)} + f_i - \dot{p}_i \right) \cdot \delta r_i = 0.$$

By restricting ourselves to a system for which the virtual work of the forces of constraint vanishes we get:

$$\sum_{i=1}^{n} \left(F_i^{(a)} - \dot{p}_i \right) \cdot \delta r_i = 0,$$

which is often called D'Alembert's principle. By now making a series of algebraic manipulations and subsitutions to this equation (more on these e.g. in Goldstein) the D'Alembert's principle becomes:

$$\sum_j \left\{ \frac{d}{dt} \left[\frac{\partial}{\partial \dot{q}_j} \left(\sum_i \frac{1}{2} m_i v_i^2 \right) \right] - \frac{\partial}{\partial q_j} \left(\sum_i \frac{1}{2} m_i v_i^2 \right) - Q_j \right\} \delta q_j = 0,$$

where

$$\dot{q}_j = \frac{dq_j}{dt} \quad \text{and} \quad Q_j = \sum_i F_i \cdot \frac{\partial r_i}{\partial q_j}.$$

The symbol Q_j is known as the generalized force in mechanics. We now recognize the two sums involving the particle masses are the total kinetic energy $T = \sum_i \frac{1}{2} m_i v_i^2$ of the system and so we can write the D'Alembert's principle as:

$$\sum_j \left\{ \frac{d}{dt}\left[\frac{\partial T}{\partial \dot{q}_j}\right] - \frac{\partial T}{\partial q_j} - Q_j \right\} \delta q_j = 0.$$

If one has experience with calculus of variation the above equations should start to look familiar. Anyway, we now remember that since the generalized coordinates q_j could be arbitrarily chosen due to independency, it follows that in order for the above equation to hold it must be that each of the terms in the sum vanish, that is:

$$\frac{d}{dt}\left[\frac{\partial T}{\partial \dot{q}_j}\right] - \frac{\partial T}{\partial q_j} - Q_j = 0 \quad \forall j.$$

Furthermore, if all the applied forces are derivable from a scalar potential energy function V, that is:

$$F_i^{(a)} = -\nabla_i V,$$

then we have that:

$$Q_j = \sum_i F_i \cdot \frac{\partial r_i}{\partial q_j} = -\sum_i \nabla_i V \cdot \frac{\partial r_i}{\partial q_j} = -\frac{\partial V}{\partial_j},$$

and also assuming that V does not depend on time t and the generalized velocities $\dot{q}_j$ (an assumption perfectly valid in many real applications) we get that the individual terms in the sum get the form:

$$\frac{d}{dt}\left[\frac{\partial (T-V)}{\partial \dot{q}_j}\right] - \frac{\partial (T-V)}{\partial q_j} = 0 \quad \forall j,$$

or that

$$\boxed{\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_j}\right) - \frac{\partial L}{\partial q_j} = 0 \quad \forall j,}$$

which is referred to as the **Lagrange's equations** of motion with the Lagrangian function $L = T - U$, that the difference of kinetic and potential energy of the system.

So what have achieved now? We have transformed the equations describing the mechanics of the system into a different yet equivalent format expressed explicitly in terms of energies of the system. In the Newtonian equations, we were dealing with complicated equations of forces et cetera. Now, we are expressing the same dynamics of the system with energies. We thus only need to know the Lagrangian of the physical system we are interested about and we can then use the Lagrange's equations of motion to find out its dynamics. In many cases in physics, this makes the imvestigation of the system's mechanics a lot easier. There is also another way of producing Lagrange's equations using the variational approach, but we will not cover this in this tutorial and it can be found in many books on mechanics (e.g. Goldstein).

Example of Lagrangian mechanics: Atwood machine

We will now go through the same Atwood machine example as we did with Newtonian mechanics. Recall that in the Lagrangian mechanics, we approach the problem by considering system energies via the Lagrangian function. So, lets find it out, we need to find the kinetic energy T and the potential energy V of the system. Note also that the Lagrangian formulation relies on conservative forces, a property valid in many physical applications (e.g. gravity, electromagnetism).

First of all, we must recognize the coordinates of the system. We denote x to be the distance from the pulley to object with mass m_1. The corresponding distance for object 2 is $l - x$, where l is the length of the inextensible string connecting the two objects. We assume the height from the ground to the pulley is h. Since the string is inextensible we see that the only variable coordinate in this system is x, that is x determines the position of objects 1 and 2 at all time.

So what are holonomic constraint equations? What are the generalized coordinates? We denote the height (position coordinate) of object 1 from ground as $h - x$ and the height of object 2 from the ground as $h - (l - x)$. The constraint we have here for the position vectors $r_1 = x, r_2 = l - x$ is that the length of the string must always be l, that is:

$$r_1 + r_2 = x + l - x = l,$$

which is trivially the case and so the only 'generalized' coordinate is x. Next, we notice that the kinetic energy of the Atwood machine is given by:

$$T = \frac{1}{2}m_1\dot{x}^2 + \frac{1}{2}m_2\dot{x}^2,$$

where $\dot{x} = \frac{dx}{dt}$ is the velocity of the 'object-string-object' system. The potential energy of the system is given by:

$$V = m_1 g(h - x) + m_2 g(h - (l - x)),$$

and so the Lagrangian function is:

$$L(x, \dot{x}) = T(x) - V(x) = \frac{1}{2}m_1\dot{x}^2 + \frac{1}{2}m_2\dot{x}^2 - m_1 g(h - x) - m_2 g(h - (l - x)),$$

or

$$L(x, \dot{x}) = \frac{1}{2}(m_1 + m_2)\dot{x}^2 + g(m_1 - m_2)x + C,$$

where C is a constant value. We now recognize the components needed for the Lagrangian equations as:

$$\frac{\partial L}{\partial \dot{x}} = (m_1 + m_2)\dot{x} \quad \text{and} \quad \frac{\partial L}{\partial x} = g(m_1 - m_2),$$

and so we get from Lagrangian equation:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right) - \frac{\partial L}{\partial x} = \frac{d}{dt}((m_1 + m_2)\dot{x}) - g(m_1 - m_2) = (m_1 + m_2)\ddot{x} - g(m_1 - m_2) =$$

that is:

$$\ddot{x} = a = g\frac{m_1 - m_2}{m_1 + m_2},$$

which is as expected the exactly same result we got with Newtonian mechanics. This particular example does not highlight the benefits of Lagrangian mechanics but it is much easier to use Lagrangian than Newtonian mechanics in more complicated applications. The point of this example is to simply illustrate how we can derive the same equations of motion for Atwood machine knowing only the Lagrangian of the system energies.

4.3. HAMILTONIAN MECHANICS

We still have one more (and the key) concept to deal with in addition to Newtonian and Lagrangian mechanis, which is the Hamiltonian way of dealing with mechanics. Hamiltonian mechanics also lays foundations for the formulations of statistical and quantum mechanics. Like before, the Hamiltonian formulation of mechanics is equivalent with the two previous one but now instead of expressing the energy function as a function of generalized coordinates and velocities and time $(q, \dot{q}, t)$, we wish to express the system energy in terms of generalized coordinates, momentum and time (q, p, t). That said, we now define the generalized or conjugate momentum as:

$$p_i = \frac{\partial L(q, \dot{q}, t)}{\partial \dot{q}_i}.$$

Furthermore, substituting this definition into the Lagrange's equations we get:

$$\frac{dp_i}{dt} - \frac{\partial L}{\partial q_j} = 0 \quad \rightarrow \quad \dot{p}_i = \frac{\partial L}{\partial q_j}.$$

In order to produce the new energy function (called the Hamiltonian), we apply the Legendre transformation for the Lagrangian, which is tailored for just this type of change of variables. Consider a function of only two variables $f(x, y)$, so that a differential of f has the form:

$$df = u\,dx + v\,dy,$$

where

$$u = \frac{\partial f}{\partial x}, \quad v = \frac{\partial f}{\partial y}.$$

We want now to change the basis of description from x, y t a new set of variables u, y, so that differential quantities are expressed in terms of the differentials du and dy. Let g be a fucntion of u and y defined as:

$$g(u, y) = f(x, y) - ux.$$

A differential of g is then given as:

$$dg = df - u\,dx - x\,du = u\,dx + v\,dy - u\,dx - x\,du = v\,dy - x\,du$$

which is now exactly in the form desired. The quantities x and v are now functions of the variables u and y given by:

$$x = -\frac{\partial g(u, y)}{\partial u}, \quad v = \frac{\partial g(u, y)}{\partial y}.$$

Next, we define the negative of the Hamiltonian function as:

$$-H(q,p,t) = L(q,\dot{q},t) - \sum_j \dot{q}_j p_j$$

and we apply the Legendre transformation to multivariable case. First, we have the differential of the Lagrangian:

$$dL = dL(q_1,\ldots,q_n,\dot{q}_1,\ldots,\dot{q}_n,t) = \sum_j \dot{p}_j\, dq_j + \sum_j p_j\, d\dot{q}_j + \frac{\partial L}{\partial t}\, dt,$$

where

$$\dot{p}_j = \frac{\partial L}{\partial q_j}, \quad p_j = \frac{\partial L}{\partial \dot{q}_j}.$$

We now use the Legendre transformation to get the negative of the Hamiltonian as (I use negative notation to be consistent with the Legendre transformation above):

$$-H(q_1,\ldots,q_n,p_1,\ldots,p_n,t) = L(q_1,\ldots,q_n,\dot{q}_1,\ldots,\dot{q}_n,t) - \sum_j p_j\dot{q}_j,$$

and the differential of this is:

$$-dH = dL - \sum_j p_j\, d\dot{q}_j - \sum_j \dot{q}_j\, dp_j,$$

or

$$-dH = \sum_j \dot{p}_j\, dq_j + \sum_j p_j\, d\dot{q}_j - \sum_j p_j\, d\dot{q}_j - \sum_j \dot{q}_j\, dp_j + \frac{\partial L}{\partial t}\, dt = \sum_j \dot{p}_j\, dq_j - \sum$$

Also, since we know that $-H = -H(q_1,\ldots,q_n,p_1,\ldots,p_n,t)$ we get the differential as:

$$-dH = \sum_j \frac{\partial(-H)}{\partial q_j}\, dq_j + \sum_j \frac{\partial(-H)}{\partial p_j}\, dp_j + \frac{\partial L}{\partial t}\, dt.$$

Comparing the two different differential expressions for $-H$, we see that:

$$\dot{p}_j = \frac{\partial(-H)}{\partial q_j} \Rightarrow -\dot{p}_j = \frac{\partial H}{\partial q_j}$$
$$-\dot{q}_j = \frac{\partial(-H)}{\partial p_j} \Rightarrow \dot{q}_j = \frac{\partial H}{\partial p_j}.$$

Also, from the partial derivative of the Lagrangian w.r.t time:

$$\frac{\partial L}{\partial t} = \sum_j \frac{\partial L}{\partial q_j}\frac{\partial q_j}{\partial t} + \sum_j \frac{\partial L}{\partial \dot{q}_j}\frac{\partial \dot{q}_j}{\partial t} + \frac{\partial L}{\partial t} = \sum_j \dot{p}_j\,\dot{q}_j + \sum_j p_j\,\ddot{q}_j + \frac{\partial L}{\partial t}$$

we see that

$$\frac{\partial(-H)}{\partial t} = -\frac{\partial H}{\partial t} = \frac{\partial L}{\partial t} - \sum_j \frac{\partial p_j}{\partial t}\,\dot{q}_j - \sum_j p_j\,\frac{\partial \dot{q}_j}{\partial t} = \sum_j \dot{p}_j\,\dot{q}_j + \sum_j p_j\,\ddot{q}_j + \frac{\partial L}{\partial t} -$$

Thus we have derived the (canonical) **Hamiltonian equations** of motion:

$$\boxed{\begin{aligned} -\dot{p}_j &= \frac{\partial H}{\partial q_j} \\ \dot{q}_j &= \frac{\partial H}{\partial p_j} \qquad \forall j \\ -\frac{\partial L}{\partial t} &= \frac{\partial H}{\partial t} \end{aligned}}$$

Few last words about the Hamiltonian energy function. Recall that in the Lagrangian mechanics we defined the Lagrangina function as the difference between system kinetic and potential energy $L = T - V$. What is the case with the Hamiltonian H? Lets take a closer look at this. For a very wide range of systems and sets of generalized coordinates, the Lagrangian can be decomposed as regards its functional behavior in the $\dot{q}$ variables as:

$$L(q, \dot{q}, t) = L_0(q, t) + L_1(q, \dot{q}, t) + L_2(q, \dot{q}, t),$$

where L_2 is a homogeneous function of the second degree (not merely quadratic) in $\dot{q}$, while L_1 is homogeneous of the first degree in $\dot{q}$. Recall that the Euler's theorem states that if a function $f(x_1, x_2, \dots)$ is a homogeneous function of degree m in the variables x_j, then:

$$\sum_j x_j \frac{\partial f}{\partial x_j} = nf.$$

There is no reason intrinsic to mechanics that requires the Lagrangian to conform to the above decomposition, but in fact it does for most problems of interest. The Lagrangian has this form when the forces are derivable from a potential not involving the velocities.

With these in mind, we apply the Euler's theorem and the Lagrangian decomposition to the Hamiltonian to get:

$$H = \sum_j \dot{q}_j p_j - L = \sum_j \dot{q}_j \frac{\partial L}{\partial \dot{q}_j} - L = \sum_j \dot{q}_j \frac{\partial L_0}{\partial \dot{q}_j} + \sum_j \dot{q}_j \frac{\partial L_1}{\partial \dot{q}_j} + \sum_j \dot{q}_j \frac{\partial L_2}{\partial \dot{q}_j} - L_0$$

If further the transformation equations defining the generalized coordinates are independent of time t and the potential energy of the system is independent of the generalized velocities $\dot{q}$, then we have that $L_2 = T$ and $L_0 = -V$ and so the Hamiltonian is:

$$H = L_2 - L_0 = T - (-V) = T + V = E,$$

where E is the total mechanical energy of the system. To summarize the mechanics we have been going through, the following table illustrates the main concepts between the three approaches to mechanics:

Mechanics	Newtonian	Lagrangian
Fundamental concepts	System forces, force diagrams constraints	System energies
Variables of interest	position coordinates r, velocities v	generalized coordinates q, generalized velocities v

Example of Hamiltonian mechanics: Atwood machine

Let us now again calculate the equations of motion for the Atwood machine, but this time using Hamiltonian mechanics. We recall from earlier that the Lagrangian for this system was given by:

$$L(x,\dot{x}) = \frac{1}{2}(m_1+m_2)\dot{x}^2 + g(m_1-m_2)x + C.$$

The generalized momentum in this case is:

$$p = \frac{\partial L}{\partial \dot{x}} = (m_1+m_2)\dot{x} \quad \Rightarrow \quad \dot{x} = \frac{p}{m_1+m_2},$$

and so the Hamiltonian for this conservative (time independent) problem is:

$$H(x,p) = \dot{x}p - L(x,\dot{x}) = \frac{p^2}{m_1+m_2} - \frac{1}{2}(m_1+m_2)\left(\frac{p}{m_1+m_2}\right)^2 - g(m_1-m_2)x -$$

that is the Hamiltonian is:

$$H(x,p) = \frac{1}{2}\frac{p^2}{m_1+m_2} - g(m_1-m_2)x + C.$$

Applying the Hamiltonian equations of motion we get:

$$\dot{p} = -\frac{\partial H}{\partial x} = g(m_1-m_2) \quad \text{and} \quad \dot{x} = \frac{\partial H}{\partial p} = \frac{p}{m_1+m_2}.$$

We take the time derivative of the second equation to get:

$$\frac{\partial \dot{x}}{\partial t} = \ddot{x} = \frac{\dot{p}}{m_1+m_2},$$

and plugging in the expression for $\dot{p}$ we get again that:

$$a = \ddot{x} = \frac{\dot{p}}{m_1+m_2} = g\frac{m_1-m_2}{m_1+m_2},$$

as expected.

7. Markov chains

7.1 Stochastic processes

We're now going to talk a little about stochastic processes. So what are stochastic processes and why are they relevant? Stochastic processes originate from physics, where in many applications an investigator is interested on seemingly random processes. Think of a dust particle which you can see on a sunny day on your house. The movement of the dust particles seems to be almost randomly governed and you can think of them as stochastic (that is, random) processes. This kind of investigation of random processes leads fast to statistical mechanics and quantum physics.

Anyway, lets get our hands dirty and make some definitions we need.

A Stochastic process is a family of ordered random variables X_t, where t ranges over some suitable index set I, e.g. $I = 1, 2, \ldots$. So you can think of the random movement of a dust particle at times $t_1, t_2, t_3, \ldots$ described by the corresponding random variables $X_1, X_2, X_3, \ldots$. Obviously these random variables are dependent on each other (at least with the dust particle case), but in many cases the state of random variable X_t is dependent only on the previous variable X_{t-1} and not on the ones before time step $t - 1$. This kind of stochastic processes are called Markov chains and we will give them a more accurate definition next:

A discrete time stochastic process $\{X_t\}_{t=0}^{\infty}$ is called a Markov chain (MC) if for all $t \geq 0$ and for all $i_0, i_1, \ldots, i_{t-1}, i, j$ we have:

$$P(X_{t+1} = j \mid X_t = i, X_{t-1} = i_{t-1}, \ldots, X_0 = i_0) = P(X_{t+1} = j \mid X_t = i).$$

If the above equation holds for any t, then we call the $\{X_t\}_{t=0}^{\infty}$ a homogeneous MC. MC has the nice property that its time evolving distribution can be completely specified by a transition probability matrix $\mathcal{P}$, where the ijth (row, column) component pf this matrix is defines as:

$$p_{ij} = P(X_1 = j \mid X_0 = i), \quad \sum_j p_{ij} = 1, p_{ij} \geq 0 \ \forall \ i, j,$$

that p_{ij} describes the probability that the chain at time $t = 1$ obtains state j when the state at time $t = 0$ is i. The above was called a 1-step transition probability and we straight away generalize this to the n-step transition probability and define:

$$p_{ij}^{(n)} = P(X_n = j \mid X_0 = i), \quad \sum_j p_{ij}^{(n)} = 1, p_{ij}^{(n)} \geq 0 \ \forall \ i, j,$$

and the corresponding matrix for these n-step transition probabilities is denoted as $\mathcal{P}^{(n)}$. It is easy to show (using the law of total probability) that:

$$\mathcal{P}^{(n)} = \mathcal{P}^n = \underbrace{\mathcal{P}\mathcal{P}\ldots\mathcal{P}}_{n \text{ multiplications}}.$$

Thus, if the vector $\mathbf{v} = (v_1, v_2, \ldots, v_s)$ describes the initial probability distribution of random variable X_0, that is $v_i = P(X_0 = i)$ where s is the number of possible states, then the corresponding distributions after n steps is given by:

$$\mathbf{v}^{(n)} = (v_1^{(n)}, v_2^{(n)}, \ldots, v_s^{(n)}) = \mathbf{v}\mathcal{P}^{(n)}.$$

Chain of random variables

What is a Markov Chain? What is a stochastic process

8. Monte carlo

8.1 Simulation

So far, we have been talking about Monte Carlo simulation. As we recall, the point begind the MC method is to estimate statistics of interest via simulation, which would otherwise be very hard or impossible to calculate analytically. For example, we are

familiar that in Bayesian statistics we are many times interested in sampling from a posterior distribution, which is defined by definition as:

$$p(B|A) = \frac{p(A|B)p(B)}{p(A)} = \frac{p(A|B)p(B)}{\int p(A|B)p(B)\,dB} = \frac{p(A|B)p(B)}{\int p(A \cap B)\,dB}. \tag{8.1}$$

Now, many times in Bayesian inference the reason why we need to apply simulation methods is because of the denominator in the above equation, that is $P(A)$. This value is a constant (since A is a given constant value or set of values) but in order to explicitly solve $p(B|A)$ we need to calculate this constant. So what gives? Why don't we just calculate it? Well, it turns out that in many problems this is either very hard or computationally expensive. Just to make things concrete, lets consider a toy example.

Assume that we have observed a set of i.i.d. (independent and identically distributed) sample vectors $\mathcal{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N\}, \mathbf{y}_i \in \mathcal{R}^K$ from some random vector $\mathbf{y} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ where we assume $\boldsymbol{\mu}$ to be unknown variable and $\boldsymbol{\Sigma}$ is a known constant (for simplicity's sake). We wish to calculate some statistic of interest from the posterior distribution of $\boldsymbol{\mu}$. Let that statistic be e.g.:

$$E_{\boldsymbol{\mu}|\mathcal{Y}}\left[h(\boldsymbol{\mu})\right] = E_{\boldsymbol{\mu}|\mathcal{Y}}\left[\sum_{i=1}^{K} \mu_i\right] = \int \sum_{i=1}^{K} \mu_i \, p(\boldsymbol{\mu}|\mathcal{Y})\,d\boldsymbol{\mu}, \tag{8.2}$$

where K is the dimensionality of $\boldsymbol{\mu}$. So what is the posterior distribution for $\boldsymbol{\mu}$? It is by definition:

$$p(\boldsymbol{\mu}|\mathcal{Y}) = \frac{p(\mathcal{Y}|\boldsymbol{\mu})p(\boldsymbol{\mu})}{p(\mathcal{Y})} = \frac{p(\mathcal{Y}|\boldsymbol{\mu})p(\boldsymbol{\mu})}{\int p(\mathcal{Y}|\boldsymbol{\mu})p(\boldsymbol{\mu})\,d\boldsymbol{\mu}},$$

where I have omitted the $\boldsymbol{\Sigma}$ from the equation since it is a constant (but it is there under the hood). We need next an expression for the prior $p(\boldsymbol{\mu})$. Assume that we have a Diriclet (multivariable Beta) prior distribution for parameters $\boldsymbol{\mu}$, that is;

$$p(\boldsymbol{\mu}) = p(\mu_1, \ldots, \mu_K, \alpha_1, \ldots, \alpha_K) = \frac{\Gamma\left(\sum_{i=1}^{K} \alpha_i\right)}{\prod_{i=1}^{K} \Gamma(\alpha_i)} \prod_{i=1}^{K} \mu_i^{\alpha_i - 1},$$

where Γ is the Gamma function and $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_K)$ is some fixed hyperparameter vector for the prior. We have the prior, now we need to solve the likelihood of the observed sample data which is:

$$p(\mathcal{Y}|\boldsymbol{\mu}) = \prod_{i=1}^{N} p(\mathbf{y}_i|\boldsymbol{\mu}) = (2\pi)^{-\frac{NK}{2}} |\boldsymbol{\Sigma}|^{-\frac{N}{2}} \exp\left(-\frac{1}{2}\sum_{i=1}^{N}(\mathbf{y}_i - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{y}_i - \boldsymbol{\mu})\right),$$

and it thus follows straightforwardly that the posterior is:

$$p(\boldsymbol{\mu}|\mathcal{Y}) = \frac{(2\pi)^{-\frac{NK}{2}} |\boldsymbol{\Sigma}|^{-\frac{N}{2}} \exp\left(-\frac{1}{2}\sum_{i=1}^{N}(\mathbf{y}_i - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{y}_i - \boldsymbol{\mu})\right) \frac{\Gamma\left(\sum_{i=1}^{K} \alpha_i\right)}{\prod_{i=1}^{K} \Gamma(\alpha_i)} \prod_{i=1}^{K} \mu_i}{\int (2\pi)^{-\frac{NK}{2}} |\boldsymbol{\Sigma}|^{-\frac{N}{2}} \exp\left(-\frac{1}{2}\sum_{i=1}^{N}(\mathbf{y}_i - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{y}_i - \boldsymbol{\mu})\right) \frac{\Gamma\left(\sum_{i=1}^{K} \alpha_i\right)}{\prod_{i=1}^{K} \Gamma(\alpha_i)} \prod_{i=1}^{K} \mu_i^{\alpha}}$$

Now, take a look at the denominator distribution:

$$p(\mathcal{Y}) = \int (2\pi)^{-\frac{NK}{2}} |\boldsymbol{\Sigma}|^{-\frac{N}{2}} \exp\left(-\frac{1}{2}\sum_{i=1}^{N}(\mathbf{y}_i - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{y}_i - \boldsymbol{\mu})\right) \frac{\Gamma\left(\sum_{i=1}^{K} \alpha_i\right)}{\prod_{i=1}^{K} \Gamma(\alpha_i)} \prod_{i=1}^{K} \mu_i^{\alpha}$$

$$= \int_{\mu_1} \int_{\mu_2} \cdots \int_{\mu_K} (2\pi)^{-\frac{NK}{2}} |\boldsymbol{\Sigma}|^{-\frac{N}{2}} \exp\left(-\frac{1}{2} \sum_{i=1}^{N} (\mathbf{y}_i - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{y}_i - \boldsymbol{\mu}) \right) \frac{\Gamma\left(\sum_{i=1}^{K} \alpha_i\right)}{\prod_{i=1}^{K} \Gamma(\alpha_i)}$$

and assume further that e.g. $K = 1000$, which is easily the case with neural networks. This is a monster integral! Or at least to me, this intergal seems a pretty hairy and time consuming task to do, but the point is that **we must calculate it if we insist an explicit formula for the posterior distribution function**, though I bet there probably exists some off-the-shelf formulas to solve this integral. It thus seems that we are badly stuck now, in order to calculate the statistic of equation (8.2) we must solve the posterior distribution first. At this point, we begin to see some hint on where the motivation comes from expressing the posterior distribution as:

$$p(B|A) \propto p(A|B)P(B),$$

in Bayesian texts often. That is, we would like to forget about the normalizing constant.

As we have seen, it can be very difficult to calculate statistics of interest from the posterior distribution. In practise, statistics containing integrals are estimated many times by sampling a large number of samples from the posterior distribution and then using these to calculate the value of interest.For example the value in equation (8.2) can be estimated by:

$$E_{\boldsymbol{\mu}|\mathcal{Y}}[h(\boldsymbol{\mu})] = \int \sum_{i=1}^{K} \mu_i \, p(\boldsymbol{\mu}|\mathcal{Y}) \, d\boldsymbol{\mu} \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{K} \mu_{ij} \qquad (8.3),$$

where μ_{ij} is the jth component of the ith sample of a random vector $\boldsymbol{\mu}$. Great! We got rid of the density function! Are we saved? No, this is because of the samples $\boldsymbol{\mu}_i$, we still need the posterior distribution function in order to construct the cumulative distribution function (CDF) which we use for data sampling. What we need now, is a way to sample data points from the posterior distribution without explicitly needing to calculate the posterior distribution and its CDF.

Fortunately, there are ways which allow us to sample from the prosterior distribution even though we do not know the normalizing constant denominator. With these approaches, we are able to sample from posterior distriobutions without knowing the CDF. I will next introduce one of the methods that helps us achieve this.

8.2 Importance sampling

Importance sampling is a method which provides a way to sample from nasty looking posterior distributions. To introduce the method, notice that:

$$E_f[h(X)] = \int h(x) f(x) \, dx = \int h(x) \frac{f(x)}{g(x)} g(x) \, dx = E_g\left[h(X) \frac{f(X)}{g(X)} \right],$$

that is we can calculate the statistic of interest (w.r.t. f) using some other, perhaps simpler, distribution g from which we can sample easily. In importance sampling, we would simply generate N samples from distribution g, and then calculate the values $h(x)\frac{f(x}{g(x)}$ and average these to get the statistic of interest. Of course, this method still includes the normalizing constant of f we desprately wished to get rid of. Lets make a small alteration to the above equations and we see that:

$$E_f[h(X)] = \frac{\int h(x)\frac{f(x)}{g(x)}g(x)\,dx}{\int \frac{f(x)}{g(x)}g(x)\,dx} = \frac{\int h(x)\frac{f(x)}{g(x)}g(x)\,dx}{\int f(x)\,dx} = \int h(x)\frac{f(x)}{g(x)}g(x)\,dx.$$

In other words, we have that:

$$E_f[h(X)] \approx \frac{\frac{1}{n}\sum_{i=1}^{n} h(x_i)\frac{f(x_i)}{g(x_i)}}{\frac{1}{n}\sum_{i=1}^{n}\frac{f(x_i)}{g(x_i)}} \underset{n\to\infty}{\to} \frac{\int h(x)\frac{f(x)}{g(x)}g(x)\,dx}{\int \frac{f(x)}{g(x)}g(x)\,dx}.$$

To give an example of this, lets apply the importance sampling to the nasty example in previous section. Before continuing, lets define few constant values in order to ease the notation and save space. Define:

$$A = \int d\boldsymbol{\mu} = \int_{\mu_1}\int_{\mu_2}\cdots d\mu_1\, d\mu_2 \cdots d\mu_K,$$

$$B = (2\pi)^{-\frac{NK}{2}}|\boldsymbol{\Sigma}|^{-\frac{N}{2}}\frac{\Gamma\left(\sum_{i=1}^{K}\alpha_i\right)}{\prod_{i=1}^{K}\Gamma(\alpha_i)},$$

$$C = p(\mathcal{Y}) = \int_{\mu_1}\int_{\mu_2}\cdots\int_{\mu_K}(2\pi)^{-\frac{NK}{2}}|\boldsymbol{\Sigma}|^{-\frac{N}{2}}\exp\left(-\frac{1}{2}\sum_{i=1}^{N}(\mathbf{y}_i-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{y}_i-\boldsymbol{\mu})\right)\frac{\Gamma}{\Gamma}$$

Notice that from these three, only C is uknown. Now we can write the example posterior distribution for $p(|\mathcal{Y})$ as:

$$p(\boldsymbol{\mu}|\mathcal{Y}) = \frac{B}{C}\exp\left(-\frac{1}{2}\sum_{i=1}^{N}(\mathbf{y}_i-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{y}_i-\boldsymbol{\mu})\right)\prod_{i=1}^{K}\mu_i^{\alpha_i-1}.$$

Let us assume that the new simpler g distribution is a multivariate uniform distribution with some limits. The probability density function is:

$$g(\boldsymbol{\mu}) = \frac{1}{A},$$

where A is what we just defined above, that is the multidimensional volume of the distribution. The statistic $h(\boldsymbol{\mu})$ we wish to calculate is still the same as above. Now we plug the components into the equations of importance sampling and we get:

$$E_{\boldsymbol{\mu}|\mathcal{Y}}\left[\sum_{j=1}^{K}\mu_i\right] = E_{\boldsymbol{\mu}}\left[h(\boldsymbol{\mu})\frac{p(\boldsymbol{\mu}|\mathcal{Y})}{g(\boldsymbol{\mu})}\right] = \frac{\int h(\boldsymbol{\mu})\frac{p(\boldsymbol{\mu}|\mathcal{Y})}{g(\boldsymbol{\mu})}g(\boldsymbol{\mu})\,d\boldsymbol{\mu}}{\int \frac{p(\boldsymbol{\mu}|\mathcal{Y})}{g(\boldsymbol{\mu})}g(\boldsymbol{\mu})\,d\boldsymbol{\mu}}$$

$$= \frac{\int \left(\sum_{j=1}^{K}\mu_j\right)\frac{\frac{B}{C}\exp\left(-\frac{1}{2}\sum_{i=1}^{N}(\mathbf{y}_i-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{y}_i-\boldsymbol{\mu})\right)\prod_{i=1}^{K}\mu_i^{\alpha_i-1}}{A^{-1}}A^{-1}\,d\boldsymbol{\mu}}{\int \frac{\frac{B}{C}\exp\left(-\frac{1}{2}\sum_{i=1}^{N}(\mathbf{y}_i-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{y}_i-\boldsymbol{\mu})\right)\prod_{i=1}^{K}\mu_i^{\alpha_i-1}}{A^{-1}}A^{-1}\,d\boldsymbol{\mu}}$$

$$= \frac{\frac{B}{C}\int \left(\sum_{j=1}^{K}\mu_j\right)\exp\left(-\frac{1}{2}\sum_{i=1}^{N}(\mathbf{y}_i-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{y}_i-\boldsymbol{\mu})\right)\prod_{i=1}^{K}\mu_i^{\alpha_i-1}\,d\boldsymbol{\mu}}{\frac{B}{C}\int \exp\left(-\frac{1}{2}\sum_{i=1}^{N}(\mathbf{y}_i-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{y}_i-\boldsymbol{\mu})\right)\prod_{i=1}^{K}\mu_i^{\alpha_i-1}\,d\boldsymbol{\mu}}$$

$$= \frac{\int \left(\sum_{j=1}^{K}\mu_j\right)\exp\left(-\frac{1}{2}\sum_{i=1}^{N}(\mathbf{y}_i-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{y}_i-\boldsymbol{\mu})\right)\prod_{i=1}^{K}\mu_i^{\alpha_i-1}\,d\boldsymbol{\mu}}{\int \exp\left(-\frac{1}{2}\sum_{i=1}^{N}(\mathbf{y}_i-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{y}_i-\boldsymbol{\mu})\right)\prod_{i=1}^{K}\mu_i^{\alpha_i-1}\,d\boldsymbol{\mu}}$$

$$\approx \frac{\sum_{m=1}^{M}\left(\sum_{i=1}^{K}\mu_{mi}\right)\exp\left(-\frac{1}{2}\sum_{i=1}^{N}(\mathbf{y}_i-\boldsymbol{\mu}_m)^T\boldsymbol{\Sigma}^{-1}(\mathbf{y}_i-\boldsymbol{\mu}_m)\right)\prod_{i=1}^{K}\mu_{mi}^{\alpha_i-1}}{\sum_{m=1}^{M}\exp\left(-\frac{1}{2}\sum_{i=1}^{N}(\mathbf{y}_i-\boldsymbol{\mu}_m)^T\boldsymbol{\Sigma}^{-1}(\mathbf{y}_i-\boldsymbol{\mu}_m)\right)\prod_{i=1}^{K}\mu_{mi}^{\alpha_i-1}},$$

where M is the number of realizations from distribution $g(\boldsymbol{\mu})$ which we can sample very easily (uniform sampling). Voila! We are done! We now have way of simulating values and generating an estimate for the statistic. The above final result looks hairy, I know, but it is computable! We do not have any ugly integrals we need to solve. We simply generate random vectors from K-dimensional uniform distribution (assuming we have also $\mathcal{Y}, \boldsymbol{\Sigma}$ and $\boldsymbol{\alpha}$), plug them into the above formula and we get the estimate. To conclude, we have found that:

$$\boxed{E_{\boldsymbol{\mu}|\mathcal{Y}}\left[\sum_{j=1}^{K}\mu_i\right] \approx \frac{\sum_{m=1}^{M}\left(\sum_{i=1}^{K}\mu_{mi}\right)\exp\left(-\frac{1}{2}\sum_{i=1}^{N}(\mathbf{y}_i-\boldsymbol{\mu}_m)^T\boldsymbol{\Sigma}^{-1}(\mathbf{y}_i-\boldsymbol{\mu}_m)\right)\prod_{i=1}^{K}}{\sum_{m=1}^{M}\exp\left(-\frac{1}{2}\sum_{i=1}^{N}(\mathbf{y}_i-\boldsymbol{\mu}_m)^T\boldsymbol{\Sigma}^{-1}(\mathbf{y}_i-\boldsymbol{\mu}_m)\right)\prod_{i=1}^{K}\mu_{mi}^{\alpha_i-1}}}$$

9. Markov chain Monte Carlo (MCMC)

9.1 Sampling, convergence, ergodicity, entropy

relation between probability and disorder

Boltzmann's entropy formula in ideal gas

The Boltzmann formula shows the relationship between entropy and the number of ways the atoms or molecules of a thermodynamic system can be arranged. It is a probability equation relating the entropy S of an ideal gas to the quantity W, the number of real microstates corresponding to the gas' macrostate. Specifically, we have:

$$S = k_B \ln W,$$

where k_B is the Boltzmann constant. How surprising is an event? Informally, the lower probability you would've assigned to an event, the more surprising it is, so surprise seems to be some kind of decreasing function of probability. It's reasonable to ask that it

be continuous in the probability. And if event A has a certain amount of surprise, and event B has a certain amount of surprise, and you observe them together, and they're independent, it's reasonable that the amount of surprise adds.

From here it follows that the surprise you feel at event A happening must be a positive constant multiple of −logP(A) (exercise; this is related to the Cauchy functional equation). Taking surprise to just be −logP(A), it follows that the entropy of a random variable is its expected surprise, or in other words it measures how surprised you expect to be on average after sampling it.

Closely related is Shannon's source coding theorem, if you think of −logP(A) as a measure of how many bits you need to tell someone that A happened.

Ergodicity

Think of X with initial conditions 1, 2, 3. If all the chains end up in same path --> Ergodic process. Otherwise, non-ergodic.

In physics and thermodynamics, the ergodic hypothesis says that, over long periods of time, the time spent by a system in some region of the phase space of microstates with the same energy is proportional to the volume of this region, i.e., that all accessible microstates are equiprobable over a long period of time.

Liouville's theorem states that, for Hamiltonian systems, the local density of microstates following a particle path through phase space is constant as viewed by an observer moving with the ensemble (i.e., the convective time derivative is zero). Thus, if the microstates are uniformly distributed in phase space initially, they will remain so at all times. But Liouville's theorem does not imply that the ergodic hypothesis holds for all Hamiltonian systems.

The ergodic hypothesis is often assumed in the statistical analysis of computational physics. The analyst would assume that the average of a process parameter over time and the average over the statistical ensemble are the same. This assumption that it is as good to simulate a system over a long time as it is to make many independent realizations of the same system is not always correct.

In probability theory, an ergodic dynamical system is one that, broadly speaking, has the same behavior averaged over time as averaged over the space of all the system's states in its phase space. In physics the term implies that a system satisfies the ergodic hypothesis of thermodynamics.

A random process is ergodic if its time average is the same as its average over the probability space, known in the field of thermodynamics as its ensemble average. The state of an ergodic process after a long time is nearly independent of its initial state.[

10. Short intro on Bayesian modeling

10.1 Bayes formula

10.2 Bayesian models, predictive distribution

10.3 Expectation-maximization?...

11. Hamiltonian Monte Carlo (HMC)

Leapfrog integration

12. Training deep CNNs via HMC

12.1 Back propagantion with CNN

12.2 Constructing the Hamiltonian with CNN

13. Python implementation

Type *Markdown* and LaTeX: α^2

In [ ]:

```cpp
/*
 * NeuralNetwork.cpp
 *
 *  Created on: Feb 27, 2020
 *      Author: jonne
 */

#include "NeuralNetwork.h"
#include "dataPoint.h"
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include <cstdlib>
#include <ctime>
#include <algorithm>
#include <string>


NeuralNetwork::NeuralNetwork(DataSet dataset, int data_dimension, s
    this->dataset = dataset;
    this->number_of_layers = numbers_of_neurons.size();
    this->numbers_of_neurons = numbers_of_neurons;
    this->batch_size = batch_size;
    this->data_dimension = data_dimension;
    this->activation_type = activation_type;
    this->learning_rate = learning_rate;

    initializeNetwork();
    // TODO Auto-generated constructor stub

}

NeuralNetwork::NeuralNetwork(int){
    // TODO Auto-generated constructor stub
    std::cout << "done";
}

NeuralNetwork::~NeuralNetwork() {
    // TODO Auto-generated destructor stub
}


void NeuralNetwork::initializeNetwork() {
    std::srand(time(0)); // Setting the random seed
    std::vector<std::vector<Neuron>> neuron_network;
    for (int w = 0; w < this->number_of_layers; w++) { // This does
        std::vector<Neuron> layer_neurons;
        for (int v = 0; v < this->numbers_of_neurons[w]; v++) {
            std::vector<double> neuron_weights;
            int number_of_weights;
            if (w == 0) { // In this case we are in the first hidde
                number_of_weights = this->data_dimension + 1; // In
            }
            else { // In this case we are on second or larger hidde
                number_of_weights = this->numbers_of_neurons[w-1] +
            }
            for (int i = 0; i < number_of_weights; i++) {
                neuron_weights.push_back(((double)rand()) / (double
            }
```

```
            layer_neurons.push_back(Neuron(w, neuron_weights));
        }
        neuron_network.push_back(layer_neurons);
    }
    this->neuron_network = neuron_network;
}


// MAYBE EDIT
void NeuralNetwork::printNetwork() {
    for (int w = 0; w < this->number_of_layers; w++) { // Loop thro
        for (int v = 0; v < this->neuron_network[w].size(); v++) {
            std::vector<double> neuron_weights = neuron_network[w].
            for (int i = 0; i < neuron_weights.size(); i++) {
                std::cout << "Layer " << w << ", Neuron " << v << "
            }
        }
    }
}

// TODO
void NeuralNetwork::getErrorSum() {
    double errorSum = 0;
    for(std::vector<DataPoint>::iterator it = this->dataset.dataset
        errorSum += (*it).x - (*it).y;
    }
    std::cout << "The current error sum is: " << errorSum << std::e
}


std::vector<std::vector<std::vector<double>>> NeuralNetwork::getAct
    //std::vector<DataPoint> sample = getSample();
    std::vector<std::vector<std::vector<double>>> data_activations;
    //std::cout << "Calculating activations...";
    //std::cout << "Batch size: " << sample.size() << std::endl;
    for (int data_index = 0; data_index < sample.size(); data_index
        std::vector<std::vector<double>> network_activations;
        std::vector<double> input_layer_activations;
        input_layer_activations.push_back(1.0); // Bias term
        for (int d = 0; d < this->data_dimension; d++) { // Input v
            input_layer_activations.push_back(sample.at(data_index)
        }
        network_activations.push_back(input_layer_activations);
        for (int layer_index = 0; layer_index < this->number_of_lay
            std::vector<double> previous_layer_activations = networ
            std::vector<double> layer_activations;
            std::vector<Neuron> layer_neurons = this->neuron_networ
            if (layer_index < this->number_of_layers - 1) { // Last
                layer_activations.push_back(1.0); // Bias term
            }
            for (int neuron_index = 0; neuron_index < this->numbers
                // Now need to get weights of this neuron, and acti
                std::vector<double> weights = layer_neurons.at(neur
                double sum_input = 0.0;
                for (int weight_index = 0; weight_index < weights.s
                    sum_input += weights.at(weight_index) * previou
                }
                // TODO SIGMOID FUNCTION TODO TODO TODO TODO
                if (this->activation_type.compare("sigmoid") == 0)
                    double activation_value = 1.0 / (1 + exp(-1 * s
                    layer_activations.push_back(activation_value);
```

```
                }
                else { // Equal to linear activation
                    layer_activations.push_back(sum_input);
                }
            }
            network_activations.push_back(layer_activations);
        }
        for (int i = 0; i < network_activations.size(); i++) {
            auto layer_act = network_activations.at(i);
            //std::cout << "Layer " << i << std::endl;
            //for (int j = 0; j < layer_act.size(); j++) {
            //    std::cout << layer_act.at(j) << ", ";
            //}
            //std::cout << std::endl;
        }
        data_activations.push_back(network_activations);
        // STEP 1: Input layer
    }


    // FOR ALL DATA POINTS IN CONTAINER
    // CREATE ACTIVATIONS AND SAVE THEM TO CONTAINERS
    return data_activations;
}


std::vector<std::vector<std::vector<double>>> NeuralNetwork::getDer
    // SIGMOID g(h) = 1 / (1 + exp(-bh))
    // DERIV. g'(h) = bg(h)(1 - g(h))
    //std::vector<DataPoint> sample = getSample();
    std::vector<std::vector<std::vector<double>>> data_derivatives;
    //std::cout << "Calculating derivatives...";
    //std::cout << "Batch size: " << sample.size() << std::endl;
    for (int data_index = 0; data_index < sample.size(); data_index
        std::vector<std::vector<double>> network_activations;
        std::vector<double> input_layer_activations;
        input_layer_activations.push_back(1.0); // Bias term
        for (int d = 0; d < this->data_dimension; d++) { // Input v
            input_layer_activations.push_back(1.0); // put the inpu
        }
        network_activations.push_back(input_layer_activations);
        for (int layer_index = 0; layer_index < this->number_of_lay
            std::vector<double> previous_layer_activations = networ
            std::vector<double> layer_activations;
            std::vector<Neuron> layer_neurons = this->neuron_networ
            if (layer_index < this->number_of_layers - 1) { // Last
                layer_activations.push_back(1.0); // Bias term
            }
            for (int neuron_index = 0; neuron_index < this->numbers
                // Now need to get weights of this neuron, and acti
                std::vector<double> weights = layer_neurons.at(neur
                double sum_input = 0.0;
                for (int weight_index = 0; weight_index < weights.s
                    sum_input += weights.at(weight_index) * previou
                }
                // TODO SIGMOID FUNCTION TODO TODO TODO TODO
                if (this->activation_type.compare("sigmoid") == 0)
                    double activation_value = 1.0 / (1 + exp(-1 * s
                    activation_value *= (1.0 - activation_value);
                    layer_activations.push_back(activation_value);
                }
```

```
                else { // Equal to linear activation
                    layer_activations.push_back(sum_input);
                }
            }
            network_activations.push_back(layer_activations);
        }
        for (int i = 0; i < network_activations.size(); i++) {
            auto layer_act = network_activations.at(i);
            //std::cout << "Layer " << i << std::endl;
            //for (int j = 0; j < layer_act.size(); j++) {
            //    std::cout << layer_act.at(j) << ", ";
            //}
            //std::cout << std::endl;
        }
        data_derivatives.push_back(network_activations);
        // STEP 1: Input layer
    }


    // FOR ALL DATA POINTS IN CONTAINER
    // CREATE ACTIVATIONS AND SAVE THEM TO CONTAINERS
    return data_derivatives;
}


std::vector<DataPoint> NeuralNetwork::getSample() {
    std::vector<int> batch_indexes;
    std::vector<DataPoint> batch_sample;
    std::srand(time(0));
    int i = 0;
    while (batch_indexes.size() < this->batch_size) {
        int index = (int)(((double)rand()) / (double)RAND_MAX * thi
        if (std::find(batch_indexes.begin(), batch_indexes.end(), i
            // In this case element is found from vector and we do
        }
        else { // In this case the new element was not in vector, a
            batch_indexes.insert(batch_indexes.begin() + i, index);
            i++;
            //std::cout << "Added index " << index << " to batch li
            batch_sample.push_back(this->dataset.dataset.at(index))
        }
    }
    return batch_sample;
}

// TODO, NEED TO CALCULATE ACTIVATIONS, NEED WEIGHTS, UPDATE LAYER
void NeuralNetwork::train(int iterations) {

    std::cout << "Training network for " << iterations << " iterati
    // SIGMOID g(h) = 1 / (1 + exp(-bh))
    // DERIV. g'(h) = bg(h)(1 - g(h))
    for (int layer = 0; layer < this->numbers_of_neurons.size(); la
        std::cout << this->numbers_of_neurons[layer];
    }


    for (int iteration = 0; iteration < iterations; iteration++) {
        std::vector<DataPoint> sample = getSample();
        std::vector<std::vector<std::vector<double>>> data_activati
        std::vector<std::vector<std::vector<double>>> data_derivati
        switch (this->numbers_of_neurons.size()) {
```

```
243         case 1: { // In this case we have only hidden neuron corres
244             // LOOP THROUGH WEIGHTS
245             std::vector<double> old_weights = this->neuron_network.
246             //std::cout << this->neuron_network.at(0).at(0).getWeig
247             std::vector<double> new_weights;
248             for (int weight_index = 0; weight_index < old_weights.s
249                 double old_weight = old_weights.at(weight_index);
250                 double grad_sum = 0.0;
251                 double error = 0.0;
252                 for (int data_point_index = 0; data_point_index < d
253                     std::vector<std::vector<double>> network_activa
254                     std::vector<std::vector<double>> network_deriva
255                     //grad_sum += -2*((double)rand()) / (double)RAN
256                     grad_sum += (sample.at(data_point_index).y - ne
257                     error += (sample.at(data_point_index).y - netwo
258                 }
259                 grad_sum *= -1.0;
260                 new_weights.push_back(old_weight - this->learning_r
261                 //std::cout << "Iteration: " << iteration + 1 << ",
262                 std::cout << error << std::endl;
263             }
264             neuron_network.at(0).at(0).updateWeights(new_weights);
265             //Neuron neuron = neuron_network.at(0).at(0);
266             //neuron.updateWeights(new_weights);
267             //neuron_network.at(0)[0] = neuron;
268             break;
269         }
270         case 2: {// In this case we have one hidden layer + output
271             // STEP 1: Update output layer weights (1 neuron)
272             std::vector<double> old_weights_1 = this->neuron_networ
273             std::vector<double> new_weights_1;
274             for (int weight_index = 0; weight_index < old_weights_1
275                 double old_weight = old_weights_1.at(weight_index);
276                 double grad_sum = 0.0;
277                 double error = 0.0;
278                 for (int data_point_index = 0; data_point_index < d
279                     std::vector<std::vector<double>> network_activa
280                     std::vector<std::vector<double>> network_deriva
281                     //grad_sum += -2*((double)rand()) / (double)RAN
282                     grad_sum += (sample.at(data_point_index).y - ne
283                     error += (sample.at(data_point_index).y - netwo
284                 }
285                 grad_sum *= -1.0;
286                 new_weights_1.push_back(old_weight - this->learning
287                 //std::cout << "Iteration: " << iteration + 1 << ",
288                 //std::cout << error << std::endl;
289             }
290             neuron_network.at(1).at(0).updateWeights(new_weights_1)
291             // STEP 2: Update hidden layer weights (M neurons)
292             for (int neuron_index = 0; neuron_index < this->neuron_
293                 std::vector<double> old_weights_0 = this->neuron_ne
294                 std::vector<double> new_weights_0;
295                 for (int weight_index = 0; weight_index < old_weigh
296                     double old_weight = old_weights_0.at(weight_ind
297                     double grad_sum = 0.0;
298                     double error = 0.0;
299                     for (int data_point_index = 0; data_point_index
300                         std::vector<std::vector<double>> network_ac
301                         std::vector<std::vector<double>> network_de
302 
303                         grad_sum += (sample.at(data_point_index).y
```

```
                            * network_derivatives.at(2).at(0) * net
                            * network_activations.at(0).at(weight_i
                        //grad_sum += -2*((double)rand()) / (double
                        //grad_sum += (sample.at(data_point_index).
                        error += (sample.at(data_point_index).y - n
                    }
                    grad_sum *= -old_weights_1.at(neuron_index+1);
                    new_weights_0.push_back(old_weight - this->lear
                    //std::cout << "Iteration: " << iteration + 1 <
                    std::cout << error << std::endl;
                }
                neuron_network.at(0).at(neuron_index).updateWeights
            }
            // code


            break;
        }
        case 3: // In this case we have two hidden layers + output
            break;
        case 4: // In this case we have two hidden layers + output
            break;
        }
    }


}
```

In [ ]:

```
1
```