Lecture 4 : Gradient descent and its generalisations - I

Recap on the ingredients of a ML problem:

- the data set $\mathcal{D} = \{\vec{x}_n, \vec{y}_n\}$ (with training, validation and test partitions)
  EXPERIENCE
- a model $f(\vec{x}_n, \vec{\vartheta})$
  TASK
- a loss function $l(\mathcal{D}, f(\vec{x}_n, \vec{\vartheta}))$
  PERFORMANCE
- a Bayesian framework that relates them all

$$p(\vec{\vartheta}/\mathcal{D}) = \frac{p(\vec{\vartheta})\, p(\mathcal{D}/\vec{\vartheta})}{\int d\vec{\vartheta}'\, p(\vec{\vartheta}')\, p(\mathcal{D}/\vec{\vartheta}')}$$

The question is now how to get to $p(\vec{\vartheta}/\mathcal{D})$. In this lecture, I discuss one of the most widely used classes of methods to achieve this goal: GRADIENT DESCENT. The idea is straight-forward: iteratively adjust the parameters $\vec{\vartheta}$ in the direction where the gradient of the cost function is large and negative. The procedure therefore seeks a minimum in the parameter space. Complications arise from:

- the complexity of the parameter space
- the complexity of the loss function

4.1. Gradient descent and Newton's method

We start by introducing a simple first-order gradient descent method and by comparing it to Newton's method.

Let us call the loss function, error function

$$l(\vec{x}_n, f(\vec{x}_n, \vec{\vartheta})) = \mathbb{E}(\vec{x}_n, f(\vec{x}_n, \vec{\vartheta}))$$

This error (or energy) function can be written as

$$E(\vec{x}_n, f(\vec{x}_n, \vec{\vartheta})) = \sum_{i=1}^{N} e_n(\vec{x}_n, \vec{\vartheta}) \equiv E(\vec{\vartheta})$$

In the simplest gradient descent (GD) algorithm, we update the parameters as follows.

1 Initialise the parameters to some random value $\vec{\vartheta}_0$

2 Iteratively update the parameters according to the equations

$$\vec{v}_t = \eta_t \nabla_\vartheta E(\vec{\vartheta}_t)$$

$$\vec{\vartheta}_{t+1} = \vec{\vartheta}_t - \vec{v}_t$$

for the iteration t and the "next" iteration $t+1$

Here $\nabla_\vartheta E(\vec{\vartheta})$ is the gradient of $E(\vec{\vartheta})$ w.r.t. $\vec{\vartheta}$ and η_t is the LEARNING RATE that controls how big a step one should take in the direction of the gradient at time step t. If η_t is sufficiently small, this method will converge to a local minimum of $E(\vec{\vartheta})$. However, if η_t is small, we may need too many minimisation steps to reach the global minimum. Conversely, if η_t is too large, we may miss it. The idea is to adjust η_t dynamically with t: η_t is large for small t and decreases with increasing t (power law or exponential decay).

Let us compare simple gradient descent with the Newton's method. In Newton's method, we choose the step $\vec{v}$ for the parameters in such a way to minimise a second-order

Taylor expansion to the energy function

$$E(\vec{\vartheta}+\vec{v}) \approx E(\vec{\vartheta}) + \nabla_\vartheta E(\vec{\vartheta})\vec{v} + \frac{1}{2}\vec{v}^T H(\vec{\vartheta})\vec{v}$$

where $H(\vec{\vartheta})$ is the Hessian matrix of second derivatives. Differentiating this equation with respect to $\vec{v}$ and noting that the optimal value $\vec{v}_{opt}$ is such that $\nabla_\vartheta E(\vec{\vartheta}+\vec{v}_{opt})=0$, we get

$$0 = \nabla_\vartheta E(\vec{\vartheta}) + H(\vec{\vartheta})\,\vec{v}_{opt}$$

Rearranging this expression, one gets

$$\begin{cases} \vec{v}_t = H^{-1}(\vec{\vartheta}_t)\nabla_\vartheta E(\vec{\vartheta}_t) \\ \vec{\vartheta}_{t+1} = \vec{\vartheta}_t - \vec{v}_t \end{cases}$$

Since we have no guarantee that the Hessian matrix is well-conditioned, in almost all applications of the Newton's method one replaces the inverse of the Hessian $H^{-1}(\vec{\vartheta}_t)$ by some suitably regularised pseudo-inverse, e.g. $[H(\vartheta_t)+\varepsilon\mathbb{1}]^{-1}$ with ε a small parameter. The Newton's method has two major limitations:

1 calculating a Hessian matrix is computationally expensive
2 even if we employ first-order approximations for the Hessian matrix (called quasi-Newton methods) we must store and invert a $n\times n$ matrix, when $n\ (\sim 10^6)$ is the number of parameters.

However, the Newton's method is useful to understand how to modify and improve the simple GD method. In particular Newton's method automatically adapts the learning rate of different parameters depending on the Hessian matrix. We aim to find the singular values of the Hessian matrix, which are inversely proportional to the squares of the local curvature of the surface. In other words the Newton's method automatically adjusts the step size so that one takes larger steps in flat directions and smaller steps in steep directions.

The Newton's method also allows us to develop intuition about the role of the learning rate in GD. Let us consider the special case of using GD to find the minimum of a quadratic energy function of a single parameter ϑ. Given the current value of our parameter ϑ, we can ask what is the optimal choice of the learning rate η_{opt}, where η_{opt} is defined as the value of η that allows us to reach the minimum of the quadratic energy function in a single step. To find η_{opt}, we expand the energy function to second order around the current value

$$E(\vartheta + v) = E(\vartheta_c) + \partial_\vartheta E(\vartheta)\, v + \frac{1}{2}\partial^2_\vartheta E(\vartheta)\, v^2$$

Differentiating w.r.t. v and setting $\vartheta_{min} = \vartheta_c - v$ yields

$$\vartheta_{min} = \vartheta_c - \left[\partial_\vartheta^2 E(\vartheta)\right]^{-1} \partial_\vartheta E(\vartheta)$$

Comparing with

$$\begin{cases} \vec{v}_t = \eta_t \nabla_\vartheta E(\vec{\vartheta}_t) \\ \vec{\vartheta}_{t+1} = \vec{\vartheta}_t - \vec{v}_t \end{cases}$$

One gets

$$v_t = \left[\partial_\vartheta^2 E(\vartheta)\right]^{-1} \partial_\vartheta E(\vartheta) \quad \text{and} \quad \eta_{opt} = \left[\partial_\vartheta^2 E(\vartheta)\right]^{-1}$$

One can show that there are four qualitatively different regimes that are possible.

- $\eta < \eta_{opt}$ the GD will take multiple small steps to reach the minimum
- $\eta = \eta_{opt}$ the GD reaches the minimum in a single step
- $\eta_{opt} < \eta < 2\eta_{opt}$ the GD oscillates across both sides of the potential eventually converging to the minimum
- $\eta > 2\eta_{opt}$ the GD diverges.

The reasoning can be generalised to the multi-dimensional case. The natural multidimensional generalisation of the second derivative is the Hessian $H(\theta)$. We can always perform Singular Value Decomposition (SVD) (i.e. a rotation by an orthogonal matrix for quadratic minima where the Hessian is symmetric). If we use a single learning rate for all parameters, convergence requires that

$$\eta < \frac{2}{\lambda_{max}} \qquad \lambda_{max} \text{ is the largest singular value}$$

The convergence scales with the condition number $\kappa = \dfrac{\lambda_{max}}{\lambda_{min}}$.

The simple GD has some issues.

- GD finds local minima of the loss function. Because in ML we are often dealing with extremely rugged landscapes with many local minima, this can lead to poor performance.
- Gradients are computationally expensive to compute for large

data sets. Doing this at every GD steps becomes computationally very expensive.

- GD is very sensitive to choices of the learning rate (see above)
- GD treats all directions in the parameter space uniformly. The learning rate is the same across all directions in the parameter space. For this reason, the maximum learning rate is set by the behaviour of the steepest direction and this can significantly slow the training.
- GD is sensitive to initial conditions. Depending on where one starts, one may end up in a different (local) minimum.
- GD can take exponential time to escape saddle points, even with random initialisation.

In the following, we will introduce variants of GD that address many of these shortcomings.

4.2 Stochastic gradient descent (with mini-batches)

One of the most widely-applied variants of GD is STOCHASTIC GRADIENT DESCENT (SGD). The algorithm is stochastic and stochasticity is incorporated by approximating the gradient on a subset of the data called minibatch. The size of the minibatches is almost always much smaller than the sample size (typical sizes are 10-100 data points out of 10^6 points)

N : sample size

M : minibatch size

N/M : number of minibatches

· B_k, $k=1,\dots,N/M$ is a minibatch

In SGD, at each gradient descent step we approximate the gradient using a single minibatch B_k

$$\nabla_\vartheta E(\vec{\vartheta}) = \sum_{i=1}^{N} \nabla_\vartheta e_i(\vec{x}_i, \vec{\vartheta}) \longrightarrow \sum_{i\in B_k} \nabla_\vartheta e_i(x_i, \vec{\vartheta})$$

We then cycle over all $k=1,\dots,N/M$ minibatches one at a time, and use the minibatch approximation to the gradient to update the parameters $\vec{\vartheta}$ at every step k. A full iteration over all N data points (using ALL minibatches) is called epoch. For notational convenience, we will denote the minibatch approximation to the gradient by

$$\nabla_\vartheta E^{MB}(\vec{\vartheta}) = \sum_{i\in B_k} \nabla_\vartheta e_i(\vec{x}_i, \vec{\vartheta})$$

The SGD algorithm can therefore be rewritten as

$$\begin{cases} \vec{v}_t = \eta_t \nabla_\vartheta E^{MB}(\vec{\vartheta}) \\ \vec{\vartheta}_{t+1} = \vec{\vartheta}_t - \vec{v} \end{cases}$$

Thus, in SGD, we replace the actual gradient over the full data set at each gradient descent step by an approximation to the gradient computed using a minibatch. Benefits:

- it introduces stochasticity and decreases the chance that the optimisation algorithm gets stuck in isolated local minima;
- it speeds up the computation, as one does not have to use all the N data points to approximate the gradient.
- it prevents overfitting (empirical observation) and naturally acts as regulator.

4.3 Adding momentum (GDM)

SGD is almost always used with a "momentum" or inertia term that serves as a memory of the direction we are moving in parameter space. This is typically implemented as

$$\begin{cases} \vec{v}_t = \gamma \vec{v}_{t-1} + \eta_t \nabla_\vartheta E(\vec{\vartheta}) \\ \vec{\vartheta}_{t+1} = \vec{\vartheta}_t - \vec{v}_t \end{cases}$$

where we have introduced a momentum parameter γ, with $0 \leq \gamma \leq 1$. We have dropped the explicit notation to indicate that the gradient has to be taken over a different minibatch at each step. It follows that $\vec{v}_t$ is a running average of recently encountered gradients and $(1-\gamma)^{-1}$ sets the characteristic time scale for the memory used in the averaging procedure. Indeed

$$\Delta \vec{\vartheta}_{t+1} = \gamma \Delta \vec{\vartheta}_t - \eta_t \nabla_\vartheta E(\vec{\vartheta}_t)$$

with $\Delta \vartheta_t = \vartheta_t - \vartheta_{t-1}$

$$\Delta \vec{\vartheta}_{t+1} = \vec{\vartheta}_{t+1} - \vec{\vartheta}_t = \gamma \vec{\vartheta}_t - \gamma \vec{\vartheta}_{t-1} - \eta_t \nabla_\vartheta E(\vec{\vartheta}_t)$$

$$\vec{\vartheta}_{t+1} = (\gamma - 1)\vec{\vartheta}_t - \gamma \vec{\vartheta}_{t-1} - \eta_t \nabla_\vartheta E(\vec{\vartheta}_t)$$

Let us get more intuition about these equations. Let us consider a ball of mass m moving in a viscous medium with viscous damping coefficient μ and potential $E(\vec{w})$ where $\vec{w}$ is the ball position. The ball motion is described by the following equation

$$m \frac{d^2 \vec{w}}{dt^2} + \mu \frac{d\vec{w}}{dt} = -\nabla_w E(\vec{w})$$

We can discretise this equation

$$m \frac{\vec{w}_{t+\Delta t} - 2\vec{w}_t + \vec{w}_{t-\Delta t}}{(\Delta t)^2} + \mu \frac{\vec{w}_{t+\Delta t} - \vec{w}_t}{\Delta t} = - \nabla_w E(\vec{w})$$

Rearranging this equation, we can rewrite it as

$$\Delta \vec{w}_{t+\Delta t} = - \frac{(\Delta t)^2}{m + \mu \Delta t} \nabla_w E(\vec{w}) + \frac{m}{m+\mu \Delta t} \Delta \vec{w}_t$$

where $\Delta \vec{w}_{t+\Delta t} = \vec{w}_{t+\Delta t} - \vec{w}_t$ and $\Delta \vec{w}_t = \vec{w}_t - \vec{w}_{t-\Delta t}$.

This equation is identical to

$$\Delta \vec{\theta}_{t+1} = \gamma \Delta \vec{\theta}_t - \eta_t \nabla_\theta E(\vec{\theta}_t)$$

if we identify $\vec{w} \leftrightarrow \vec{\theta}$ and

$$\gamma = \underbrace{\frac{m}{m+\mu \Delta t}}_{\substack{\text{mass of} \\ \text{the particle}}} \qquad \eta_t = \underbrace{\frac{(\Delta t)^2}{m + \mu \Delta t}}_{\text{viscous damping}}$$

Therefore γ provides inertia. In the large viscosity/small learning rate limit, our memory time scales as $(1-\gamma)^{-1} \approx \frac{m}{\mu \Delta t}$

SGD momentum helps the gradient descent algorithm gain speed in directions with persistent but small gradients even in the presence of stochasticity, while suppressing oscillations in high curvature directions.

These beneficial properties of momentum can sometimes become even more pronounced by using a slight modification of the momentum algorithm called Nesterov Accelerated

Gradient (NAG). In the NAG algorithm, rather than calculating the gradient at the current parameters, $\nabla_\theta E(\vec{\theta}_t)$, one calculates the gradient at the expected value of the parameters given our current momentum $\nabla_\theta E(\vec{\theta}_t + \gamma \vec{v}_{t-1})$. This yields

$$\begin{cases} \vec{v}_t = \gamma \vec{v}_{t-1} + \eta_t E(\vec{\theta}_t + \gamma \vec{v}_{t-1}) \\ \vec{\theta}_{t+1} = \vec{\theta}_t - \vec{v}_t \end{cases}$$

The advantage of the NAG algorithm is that it allows for the use of a larger learning rate than GDM for the same choice of γ.