Computational Physics III
Machine Learning
WS-18/19

Distribution: 01/11/18, Due: 14/11/18

Overview

In this exercise we will overview various gradient descent methods, by visualizing and applying these techniques to some simple two-dimensional surfaces. Methods studied include ordinary gradient descent, gradient descent with momentum, **NAG**, **ADAM**, and **RMSProp**.
It is important to stress out that doing gradient descent on the surfaces is different from performing gradient descent on a loss function in Machine Learning (ML). The reason is that in ML not only do we want to find good minima but we want to find good minima that generalizes well to new data. Despite this crucial difference, we can still build intuition about gradient descent methods by applying the methods to simple surfaces.

Surfaces

We will consider three simple surfaces: a quadratic minimum of the form

$$z = ax^2 + by^2,$$

a saddle-point of the form

$$z = ax^2 - by^2,$$

and **Beale's Function**, a convex function often used to test optimization problems of the form

$$z = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.6250 - x + xy)^3.$$

Gradient Descent

Let's study different gradient descent algorithms used in machine learning and the role of hyperparameters like the learning rate. Here, we confine ourselves primarily to looking at the performance in the absence of noise.
Throughout, we denote the *parameters* by θ and the *energy function* we are trying to minimize by $E(\theta)$. We start by considering a simple gradient descent method. In this method, we will take steps in the direction of the local gradient.
Given some parameters θ, we adjust the parameters at each iteration so that

$$\theta_{t+1} = \theta_t - \eta_t \nabla_\theta E(\theta)$$

where we have introduced the *learning rate* η_t that controls how large a step we take. In general, the algorithm is extremely sensitive to the choice of η_t. If η_t is too large, then one can wildly oscillate

around minima and miss important structure at small scales.
This problem is amplified if our gradient computations are noisy and inexact (as is often the case in machine learning applications). If η_t is too small, then the learning/minimization procedure becomes extremely slow.

Gradient Descent with Momentum

One problem with gradient descent is that it has no memory of where it comes from. This can be an issue when there are many shallow minima in our landscape. If we make an analogy with a ball rolling down a hill, the lack of memory is equivalent to having has no inertia or momentum (i.e. completely overdamped dynamics). Without momentum, the ball has no kinetic energy and cannot climb out of local minima. In our case this may slow down the calculation or also preventing it to find the true minimum.
Momentum becomes especially important when we start thinking about stochastic gradient descent with noisy, stochastic estimates of the gradient where we should remember where we were coming from and not react drastically to each new update.
Inspired by this, we can add a memory or momentum term to the stochastic gradient descent term above

$$\begin{aligned} v_t &= \gamma v_t - 1 + \eta_t \nabla_\theta E(\theta_t) \\ \theta_{t+1} &= \theta_t - v_t \end{aligned}$$

with $0 \leq \gamma < 1$ called the *momentum parameter*.
When $\gamma = 0$, this reduces to ordinary gradient descent, and increasing γ increases the inertial contribution to the gradient. From the equations above, we can see that typical memory lifetimes of the gradient is given by $(1-\gamma)^{-1}$. For $\gamma = 0$ as in gradient descent, the lifetime is just one step. For $\gamma = 0.9$, we typically remember a gradient for ten steps. We will call this gradient descent with classical momentum or CM for short.
A final widely used variant of gradient descent with momentum is called the **Nesterov accelerated gradient** (**NAG**). In **NAG**, rather than calculating the gradient at the current position, one calculates the gradient at θ_t the position, momentum will carry us to, at time $t+1$, namely, $\theta_i - \gamma v_{t-1}$. Thus, the update becomes

$$v_t = \gamma v_t - 1 + \eta_t \nabla_\theta E(\theta_t - \gamma v_{t-1})$$
$$\theta_{t+1} = \theta_t - v_t$$

Experiments with Gradient Descent, CM, and NAG

Let us now look at the dependence of gradient descent on learning rate in a simple quadratic minima of the form $z = ax^2 + by^2 - 1$.
Make plots below for $\eta = 0.1, 0.5, 1, 1.01$ and $a = 1$ and $b = 10$.

1. What are the qualitatively different behaviors that arise as η is increased?
2. What does this tell us about the importance of choosing learning parameters?
3. How do these change if we change a and b above?
4. In particular how does anisotropy change the learning behavior?
5. Make similar plots for **CM** and **NAG**? How do the learning rates for these procedures compare with those for gradient descent?

Gradient Descents with the second moment

In stochastic gradient descent, with and without momentum, we still have to specify a schedule for tuning the learning rates η_t as a function of time. This presents a number of dilemmas. The learning rate is limited by the steepest direction which can change depending on where in the landscape we are. To circumvent this problem, ideally our algorithm would take large steps in shallow, flat directions and small steps in steep, narrow directions. Second-order methods accomplish this by calculating or approximating the Hessian and normalizing the learning rate by the curvature. However, this is very computationally expensive for extremely large models. Ideally, we would like to be able to adaptively change our step size to match the landscape without paying the steep computational price of calculating or approximating Hessians.
Recently, a number of methods have been introduced that accomplish this by tracking not only the gradient but also the second moment of the gradient. These methods include **AdaGrad**, **AdaDelta**,**RMS-Prop**, and **ADAM**. Here, we discuss the latter of these two as representatives of this class of algorithms.
In **RMS-prop**, in addition to keeping a running average of the first moment of the gradient, we also keep track of the second moment. The update rule for **RMS-prop** is given by

$$\begin{aligned}\mathbf{g}_t &= \nabla_\theta E(\boldsymbol{\theta})\\ \mathbf{s}_t &= \beta \mathbf{s}_{t-1} + (1-\beta)\mathbf{g}_t^2\\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \eta_t \frac{\mathbf{g}_t}{\sqrt{\mathbf{s}_t + \epsilon}}\end{aligned}$$

where β controls the averaging time of the second moment and is typically taken to be about $\beta = 0.9$, η_t is a learning rate typically chosen to be 10^{-3}, and $\epsilon \sim 10^{-8}$ is a small regularization constant to prevent divergences. It is clear from this formula that the learning rate is reduced in directions where the norm of the gradient is consistently large. This greatly speeds up the convergence by allowing us to use a larger learning rate for flat directions. A related algorithm is the **ADAM** optimizer. In **ADAM**, we keep a running average of both the first and second moment of the gradient and use this information to adaptively change the learning rate for different parameters. In addition to keeping a running average of the first and second moments of the gradient, **ADAM** performs an additional a bias correction to account for the fact that we are estimating the first two moments of the gradient using a running average (denoted by the hats in the update rule below). The update rule for **ADAM** is given by (where multiplication and division are understood to be element wise operations below)

$$\begin{aligned}\mathbf{g}_t &= \nabla_\theta E(\boldsymbol{\theta})\\ \mathbf{m}_t &= \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t\\ \mathbf{s}_t &= \beta_2 \mathbf{s}_{t-1} + (1-\beta_2)\mathbf{g}_t^2\\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \eta_t \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{s}}_t + \epsilon}}\end{aligned}$$

where β_1 and β_2 set the memory lifetime of the first and second moment and are typically take to be 0.9 and 0.99 respectively, and η and ϵ are identical to **RMSprop**

Experiments with ADAM and RMSprop

We will now use a function commonly used in optimization protocols:

$$f(x, y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy)^3.$$

This function has a global minimum at $(x, y) = (3, 0.5)$. We will use gradient descent, gradient descent with classical momentum, **NAG**, **RMSprop**, and **ADAM** to find minima starting at different initial conditions. One of the things you should experiment with is the learning rate and the number of steps, we take. Initially, we have set $N_{steps} = 10^4$ and the learning rate for **ADAM/RMSprop** to $\eta = 10^{-3}$ and the learning rate for the remaining methods to 10^{-6}.

1. Make the plots for these default values. What do you see?
2. Make a plot when the learning rate of all methods is $\eta = 10^{-6}$. How does your plot change?
3. Now set the learning rate for all algorithms to $\eta = 10^{-3}$. What goes wrong? Why?

Note: Each of the parts in the exercise carry 3-points. Please write comments in your code wherever appropriate, make a text file where you give explanation for each part and (importantly) include graphs wherever appropriate.