Nesterov's Accelerated Gradient Descent Method

Limao Chang

1 Introduction

At the core of machine and deep learning methods is optimisation, the process in which a model learns optimal parameters to minimise a relevant objective function. The basis of many of these optimisation algorithms is gradient descent (GD), an algorithm which steps along the direction of steepest descent in the objective function at each iteration. It is a simple and intuitive algorithm, but suffers from some limitations as a result, especially when we start to consider more complicated objective functions.

In this tutorial paper, we will introduce Nesterov's Accelerated Gradient Descent (AGD) method, which uses acceleration- and momentum-based concepts to overcome the limitations found in GD. Before doing so, we will cover the GD and classical momentum methods, as these are prerequisite algorithms for AGD, and also serve as algorithms we can compare against AGD.

2 General Framework

We first introduce a general optimisation framework for the algorithms discussed in this paper. The aim is to minimise[1] some objective function $f(\boldsymbol{\theta})$ with respect to its parameter(s) $\boldsymbol{\theta}$. We assume that f is differentiable, that is ∇f exists. In the context of machine and deep learning, the objective is typically the loss function, where the loss is a metric of how well a model performs on a given dataset. When we talk about a model "learning", what we mean is we are finding the parameter(s) $\boldsymbol{\theta}$ that minimise the loss $f(\boldsymbol{\theta})$. In practice, and especially with (very) deep learning models, it is not possible to find the global minima, in which case we usually settle for a good local minima instead.

We start from an initial guess $\boldsymbol{\theta}_0$ in the parameter space (for simplicity we'll assume a nonconstrained parameter space). Then we update our current guess $\boldsymbol{\theta}_t$ at iteration t by stepping in some direction to arrive at the next guess $\boldsymbol{\theta}_{t+1}$. Mathematically, we can write the general parameter update at iteration $t+1$ as

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \mathbf{v}_{t+1}. \tag{1}$$

Here $\mathbf{v}_{t+1}$ is called the *velocity vector*, and is the step we take from $\boldsymbol{\theta}_t$ to $\boldsymbol{\theta}_{t+1}$. Whether $\mathbf{v}$ is subscripted by t or $t+1$ is a matter of convention – here we choose the latter as it is computed in the $(t+1)$th iteration. We can also write the parameter update in terms of the velocity vector:

$$\mathbf{v}_{t+1} = \boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t. \tag{2}$$

To compute $\mathbf{v}_{t+1}$ in (2), it is usually more useful to write it recursively, i.e. in terms of $\mathbf{v}_t$, since if we know $\boldsymbol{\theta}_{t+1}$ then we already know $\mathbf{v}_{t+1}$.

For the optimisation algorithms we discuss, there are also two hyper-parameters of interest: the learning rate $\eta \geq 0$, which determines the step size taken at each iteration, and (for momentum-based methods) the momentum coefficient α, which determines the momentum effect size at each iteration. Typically, η is a small value on the order of 10^{-4} to 10^{-1}, and α is a value between 0 and 1 (Baughman and Liu, 1995), with a widely-used and empirically-supported value of 0.9 (Ruder, 2016). In general, these hyperparameters may be *adaptive* – that is, we have η_t and α_t instead of η and α – but for simplicity in this paper we will take them to be fixed values.

[1] Note we can turn a maximisation problem on $f(\boldsymbol{\theta})$ into a minimisation problem by taking the objective to be $-f(\boldsymbol{\theta})$.

3 Gradient Descent (GD)

We begin with a brief overview of the GD algorithm. At each iteration, we simply take a step in the direction of steepest descent based on our current position; i.e., in the $(t+1)$th iteration, this is the direction given by $-\nabla f(\boldsymbol{\theta}_t)$, where ∇f is the gradient of f. It can be shown that taking a step in the stepest descent direction decreases the function value (provided the step we take is small enough). The step taken at iteration $t+1$ is then given by

$$\mathbf{v}_{t+1} = -\eta \nabla f(\boldsymbol{\theta}_t), \tag{3}$$

where η controls the size of the step taken in the steepest descent direction. The full update for GD at iteration $t+1$ is thus

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \mathbf{v}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla f(\boldsymbol{\theta}_t). \tag{4}$$

A Python implementation of the GD algorithm is provided in Algorithm 1 in Appendix A.

4 Classical Momentum

In each iteration of GD, we only rely on the gradient value of the current iterate. If we were to devise an extension to GD, a natural idea might be to use information from previous iterates in addition to the current iterate. Indeed, one extension of GD that uses this idea is *momentum*. The idea is to build up inertia in a particular search direction, in the hopes of speeding up convergence towards a minima (Brownlee, Jason, 2021). We will see that momentum can also be useful to help overcome and skip over local minima, as well as regions with very small gradients.

This concept of momentum is fairly intuitive and is borrowed from physics. The classic example is to consider a ball rolling down a hill. As it continues rolling, it builds up more momentum in the direction it has already travelled, which helps it to roll past any plateaus (local minima) or small valleys to get to the bottom of the hill (global minima).

With classical momentum, the step taken at iteration $t+1$ is given by

$$\mathbf{v}_{t+1} = \alpha \mathbf{v}_t - \eta \nabla f(\boldsymbol{\theta}_t) = \alpha(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}) - \eta \nabla f(\boldsymbol{\theta}_t). \tag{5}$$

The first term is the momentum term, which is the step taken in the previous iteration t scaled by the momentum coefficient α. Notice that if $\alpha = 0$, this is just the GD update! The second term is the same as in GD – the steepest descent direction, scaled by the learning rate η. The full parameter update at iteration $t+1$ is then given by

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \mathbf{v}_{t+1} = \boldsymbol{\theta}_t + \underbrace{\alpha(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1})}_{\text{momentum}} - \underbrace{\eta \nabla f(\boldsymbol{\theta}_t)}_{\text{GD}}. \tag{6}$$

The intuition behind the momentum term here is that **we want to travel some distance in the previous direction we stepped**. If, in the past, we generally kept travelling in one direction, then we want to build up inertia and speed up our search in this direction – similarly to how the ball would gain momentum in one particular direction as it rolled (Brownlee, Jason, 2021).

To see this notion of a "general past search direction" more explicitly, notice that, by the recursive definition of $\mathbf{v}_{t+1}$, each iteration implicitly depends on all previous iterations of the algorithm through $\mathbf{v}_t$. Since $\alpha \in [0, 1]$, recent search directions contribute more to $\alpha \mathbf{v}_t$ than past search directions. Indeed, by expanding the recursive term $\alpha \mathbf{v}_t$, we can see that it is an

exponentially weighted average of past search directions. The following exercise makes this idea explicit.

A Python implementation of the classical momentum algorithm is provided in Algorithm 2 in Appendix A.

Exercise 4.1. Using the expression for $\mathbf{v}_{t+1}$ in (5), show that $\mathbf{v}_{t+1}$ can be written as

$$\mathbf{v}_{t+1} = -\eta \sum_{i=0}^{t} \alpha^{t-i} \nabla f(\boldsymbol{\theta}_i),$$

that is, the step taken at each iteration is an exponentially weighted average of all previous GD steps. You may take $\mathbf{v}_0 = \mathbf{0}$, since at the first step we have no "memory" of any momentum.

5 Nesterov's Accelerated Gradient Descent (AGD)

We are now ready to look at Nesterov's AGD method. We present two derivations of this method, and offer interpretations for each. We claim for now that AGD can be thought of as consisting of a momentum step followed by a gradient step, and show how we can arrive at this conclusion by starting from the original AGD formulation.

5.1 Original Nesterov AGD Method

The original AGD method was proposed by Yurii Nesterov in his paper on gradient descent for minimising composite functions (Nesterov, 2013). For this derivation, it is helpful to think of each iteration as two stages: the GD stage and the "momentum-like" stage (Melville, 2016). We denote by $\boldsymbol{\phi}$ the result of the GD stage, so that $\boldsymbol{\phi}_{t+1}$ is the result of the GD stage at the $(t+1)$th iteration. Then the full AGD parameter update for one iteration is given by

$$\begin{aligned} \boldsymbol{\phi}_{t+1} &= \boldsymbol{\theta}_t - \eta \nabla f(\boldsymbol{\theta}_t), && \text{(GD stage)} \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\phi}_{t+1} + \alpha(\boldsymbol{\phi}_{t+1} - \boldsymbol{\phi}_t). && \text{(momentum-like stage)} \end{aligned} \tag{7}$$

Though the update itself appears simple, the intuition is a bit hard to grasp. We will derive an alternative expression for $\boldsymbol{\theta}_{t+1}$ that makes the link to momentum more explicit. On expanding $\boldsymbol{\phi}_{t+1}$ in (7), we have

$$\begin{aligned} \boldsymbol{\theta}_{t+1} &= (\boldsymbol{\theta}_t - \eta \nabla f(\boldsymbol{\theta}_t)) + \alpha[(\boldsymbol{\theta}_t - \eta \nabla f(\boldsymbol{\theta}_t)) - (\boldsymbol{\theta}_{t-1} - \eta \nabla f(\boldsymbol{\theta}_{t-1}))] \\ &= \boldsymbol{\theta}_t - \eta \nabla f(\boldsymbol{\theta}_t) + \alpha(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}) - \alpha(\eta \nabla f(\boldsymbol{\theta}_t) - \eta \nabla f(\boldsymbol{\theta}_{t-1})). \\ \implies \mathbf{v}_{t+1} &= \underbrace{-\eta \nabla f(\boldsymbol{\theta}_t)}_{\text{GD}} + \underbrace{\alpha(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1})}_{\text{momentum}} - \underbrace{\alpha(\eta \nabla f(\boldsymbol{\theta}_t) - \eta \nabla f(\boldsymbol{\theta}_{t-1}))}_{\text{gradient momentum}}. \end{aligned} \tag{8}$$

We can now get an idea of what the AGD update consists of. The first part of this update, labelled GD and momentum above, is exactly the update for classical momentum. Then the difference between classical momentum and AGD is that there is another momentum step, but in terms of the gradient iterates $\nabla f(\boldsymbol{\theta}_t)$ and $\nabla f(\boldsymbol{\theta}_{t-1})$ instead, which we will call "gradient momentum".

However, while this is an interesting decomposition of the parameter update, it is perhaps still a bit unclear what we mean by AGD consisting of a momentum step followed by a GD step. To resolve this, we turn to a different derivation of Nesterov's AGD method.

5.2 Sutskever's Derivation

Sutskever presents an alternative derivation of Nesterov's AGD (Sutskever et al., 2013). We shift our perspective to consider the momentum-like stage $\boldsymbol{\theta}$ as the first stage and the GD stage $\boldsymbol{\phi}$ as the second stage. This essentially amounts to following the same steps as in the NAG method, except we offset what we consider to be the end of an iteration by one stage (Melville, 2016). To illustrate this more explicitly, consider the two following AGD steps:

$$\begin{aligned}
\boldsymbol{\phi}_{t+1} &= \boldsymbol{\theta}_t - \eta \nabla f(\boldsymbol{\theta}_t) \\
\boldsymbol{\theta}_{t+1} &= \boldsymbol{\phi}_{t+1} + \alpha(\boldsymbol{\phi}_{t+1} - \boldsymbol{\phi}_t) \\
\boldsymbol{\phi}_{t+2} &= \boldsymbol{\theta}_{t+1} - \eta \nabla f(\boldsymbol{\theta}_{t+1}) \\
\boldsymbol{\theta}_{t+2} &= \boldsymbol{\phi}_{t+2} + \alpha(\boldsymbol{\phi}_{t+2} - \boldsymbol{\phi}_{t+1})
\end{aligned}$$

In Sutskever's derivation, the computation of $\boldsymbol{\theta}_{t+1}$ and $\boldsymbol{\phi}_{t+2}$ would form one iteration:

$$\begin{aligned}
\boldsymbol{\theta}_{t+1} &= \boldsymbol{\phi}_{t+1} + \alpha(\boldsymbol{\phi}_{t+1} - \boldsymbol{\phi}_t) \\
\boldsymbol{\phi}_{t+2} &= \boldsymbol{\theta}_{t+1} - \eta \nabla f(\boldsymbol{\theta}_{t+1})
\end{aligned}$$

Now that these stages belong to the same iteration, we need to relabel the iteration indices:

$$\begin{aligned}
\boldsymbol{\theta}_{t+1} &= \boldsymbol{\phi}_t + \alpha(\boldsymbol{\phi}_t - \boldsymbol{\phi}_{t-1}) \\
\boldsymbol{\phi}_{t+1} &= \boldsymbol{\theta}_{t+1} - \eta \nabla f(\boldsymbol{\theta}_{t+1})
\end{aligned}$$

Next, we can expand $\boldsymbol{\theta}_{t+1}$ in the expression for $\boldsymbol{\phi}_{t+1}$ to obtain the expression for a full parameter update in a single line. We now also note that, since $\boldsymbol{\phi}$ is the result of each iteration, we relabel $\mathbf{v}$ to be in terms of $\boldsymbol{\phi}$ instead; that is, $\mathbf{v}_{t+1} = \boldsymbol{\phi}_{t+1} - \boldsymbol{\phi}_t$ in the $(t+1)$th iteration. We have

$$\begin{aligned}
\boldsymbol{\phi}_{t+1} &= \boldsymbol{\phi}_t + \alpha(\boldsymbol{\phi}_t - \boldsymbol{\phi}_{t-1}) - \eta \nabla f(\boldsymbol{\phi}_t + \alpha(\boldsymbol{\phi}_t - \boldsymbol{\phi}_{t-1})) \\
&= \boldsymbol{\phi}_t + \alpha \mathbf{v}_t - \eta \nabla f(\boldsymbol{\phi}_t + \alpha \mathbf{v}_t).
\end{aligned}$$

Now that we have shifted our end of interaction by one stage, the very first GD step $\boldsymbol{\phi}_1 = \boldsymbol{\theta}_0 - \eta \nabla f(\boldsymbol{\theta}_0)$ is missing from this set of updates. However, this isn't really an issue, since $\boldsymbol{\theta}_0$ is arbitrarily chosen anyway – if we'd like, we could think of this starting point as a GD step from some other starting point instead.

For the sake of consistency and for ease of comparison with the original AGD derivation, let us relabel $\boldsymbol{\phi}$ to $\boldsymbol{\theta}$ now that we are done with the derivation:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \mathbf{v}_t - \eta \nabla f(\boldsymbol{\theta}_t + \alpha \mathbf{v}_t) \tag{9}$$

It is now clear what we mean when we say that AGD is a momentum step followed by a GD step – momentum is first applied to the current iterate $\boldsymbol{\theta}_t$ via $\boldsymbol{\theta}_t + \alpha \mathbf{v}_t$, and then we take a step in the steepest descent direction from this point as opposed to $\boldsymbol{\theta}_t$. Similarly to classical momentum, if $\alpha = 0$, this is simply the GD update. Now the velocity vector at the $(t+1)$th iteration can be written as

$$\mathbf{v}_{t+1} = \alpha \mathbf{v}_t - \eta \nabla f(\boldsymbol{\theta}_t + \alpha \mathbf{v}_t) = \alpha(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}) - \eta \nabla f(\boldsymbol{\theta}_t + \alpha(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1})). \tag{10}$$

Compare this with the velocity vector for classical momentum in (5).

A Python implementation of the Nesterov AGD algorithm (using Sutskever's formulation) is provided in Algorithm 3 in Appendix A.

As an aside, there is *another* popular derivation of AGD by Bengio (Bengio, Boulanger-Lewandowski, and Pascanu, 2013). According to Melville (Melville, 2016), this formulation is

perhaps a bit easier to integrate into existing software. However, the derivation is a bit more involved and so we do not delve into it here.

Exercise 5.1. Using the expression for $\mathbf{v}_{t+1}$ in (8) (that is, the velocity vector from the original AGD derivation), show that $\mathbf{v}_{t+1}$ can be written as

$$\mathbf{v}_{t+1} = -\eta \nabla f(\boldsymbol{\theta}_t) - \eta \sum_{i=0}^{t} \alpha^{t+1-i} \nabla f(\boldsymbol{\theta}_i).$$

As with Exercise 4.1, you may take $\mathbf{v}_0 = \mathbf{0}$, so that

$$\mathbf{v}_1 = -\eta \nabla f(\boldsymbol{\theta}_0) + \alpha \mathbf{v}_0 - \alpha\eta \nabla f(\boldsymbol{\theta}_0) = -(1+\alpha)\eta \nabla f(\boldsymbol{\theta}_0).$$

6 Comparison of Classical Momentum and Nesterov's AGD

Classical momentum and Nesterov's AGD are similar in that they are both momentum-based extensions of gradient descent. In this section we discuss the differences between the two methods. Recall the parameter updates for classical momentum and Nesterov's AGD:

$$\begin{aligned} \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}) - \eta \nabla f(\boldsymbol{\theta}_t) && \text{(momentum)} \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}) - \eta \nabla f(\boldsymbol{\theta}_t + \alpha(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1})) && \text{(AGD)} \end{aligned}$$

As we remarked earlier, the two updates are pretty similar. The difference is in when we take the step in the steepest descent direction. In classical momentum, we can think of taking both the momentum and gradient steps at the same time and summing the direction vectors, since we take the gradient step at the iterate $\boldsymbol{\theta}_t$. In Nesterov's AGD, the gradient step **only happens after the momentum step**, that is we step in the steepest descent direction at the point $\boldsymbol{\theta}_t + \alpha(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1})$.

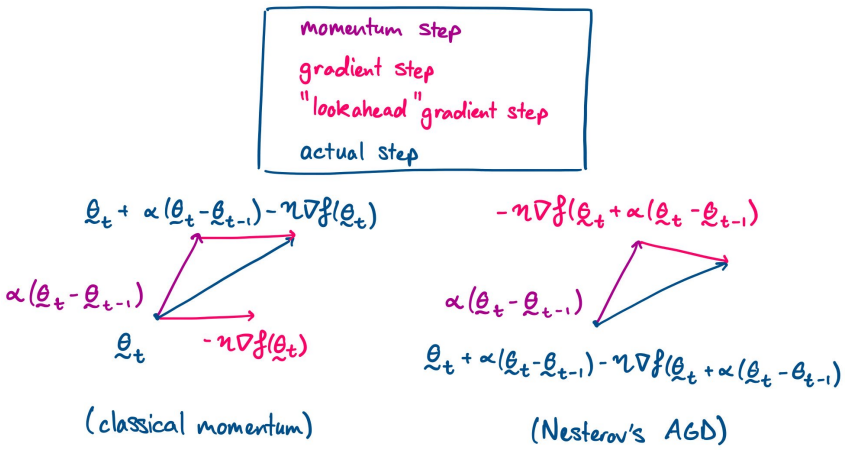

Figure 1: Classical Momentum vs Nesterov's AGD Parameter Update

The difference can be seen visually in Figure 1. For this reason, the gradient step in AGD is sometimes called a "lookahead" gradient step – we "look ahead" to the momentum step, and *then* take a step in the steepest descent direction. Computing the gradient at a point that is slightly

in the direction of momentum can help with reducing overshooting and oscillatory behaviour in the GD trajectory, e.g. in regions with high/sharp curvature. In practice, this generally leads to faster convergence and better generalisation (Bengio, Boulanger-Lewandowski, and Pascanu, 2013).

7 Discussion of GD Limitations

GD is a simple and intuitive algorithm, but also has several issues that mainly stem from it only using gradient information at each step. We will show some of these problems and illustrate how momentum-based methods like classical momentum and Nesterov's AGD can overcome them. First we consider the Rosenbrock function, which is given by

$$f(x, y) = (a - x)^2 + b(y - x^2)^2$$

for constants a, b. This is a non-convex function typically used for benchmarking optimisation algorithms. It is characterised by a long and narrow valley with a global minimum at $(x, y) = (a, a^2)$. This function is difficult for GD methods to optimise, due to the valley's elongated shape. Its gradient diminishes very quickly as we move away from the global minimum along the valley. This is a problem for GD, as it means the steps taken will be too small for the algorithm to converge to the global minimum.

We demonstrate this by running GD, classical momentum, and AGD with hyperparameters $\eta = 0.00015$ and $\alpha = 0.9$ on the function with $a = 1, b = 100$. The trajectory of the three algorithms is plotted against the contour plot of f in Figure 2 below.

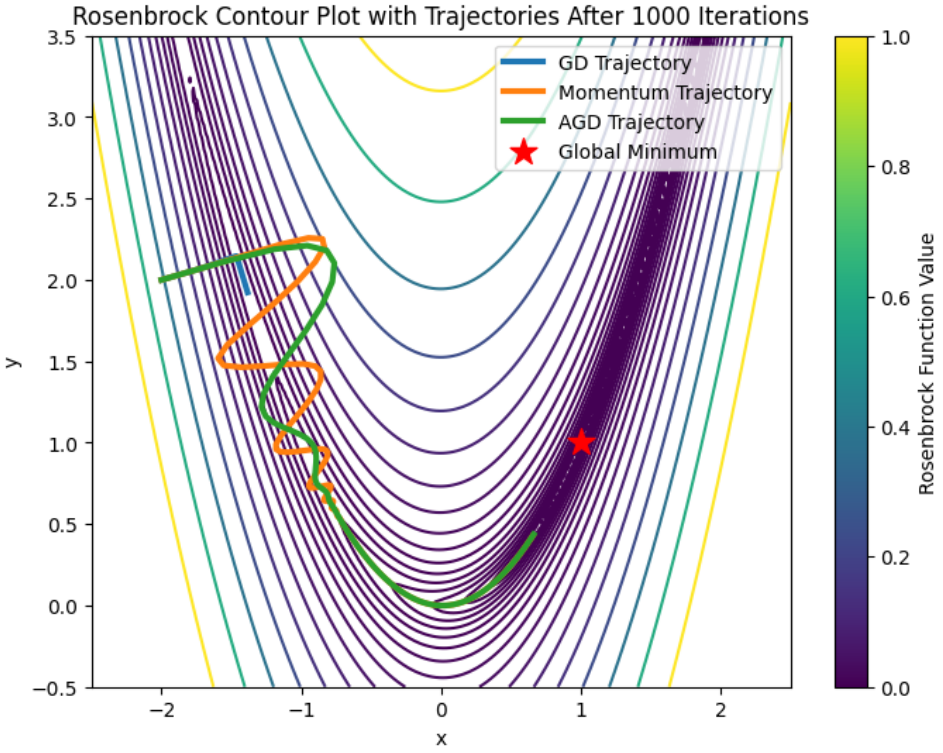

Figure 2: GD, Classical Momentum, and AGD on $f(x, y) = (1 - x)^2 + 400(y - x^2)^2$

Indeed, we see that the GD trajectory gets stuck shortly after it enters the valley – the gradient values are too small for it to be able to make any more progress. However, the

momentum and AGD trajectories oscillate back and forth between the valley, making it less likely to get stuck in regions with very small gradients. While both algorithms eventually fall into the valley, their respective momentum mechanisms allow them to speed up their search in the direction of the global minimum, and they are able to still make progress towards it.

Another issue that can arise when using GD is when the objective is non-convex and has more than one local minima. Generally, there is no way to tell whether we are at a global or local minima based on gradient information alone. This can be problematic for convergence and efficiency, since ∇f will (similarly to the previous situation) be near zero around local minima, resulting in insignificant parameter updates. To demonstrate this, consider applying the three algorithms to the objective $f(x) = x^4 - 3x^3 + 2x^2 + x$ with starting points $x_0 = -0.5$ and $x_0 = 2.0$ in Figure 3 below. All three algorithms used a learning rate of $\eta = 0.01$, and were halted early if $\|\nabla f(x_t)\| < 0.0001$ with a maximum of 500 iterations. The final iterate and number of iterations to convergence for each run are tabulated in Table 1.

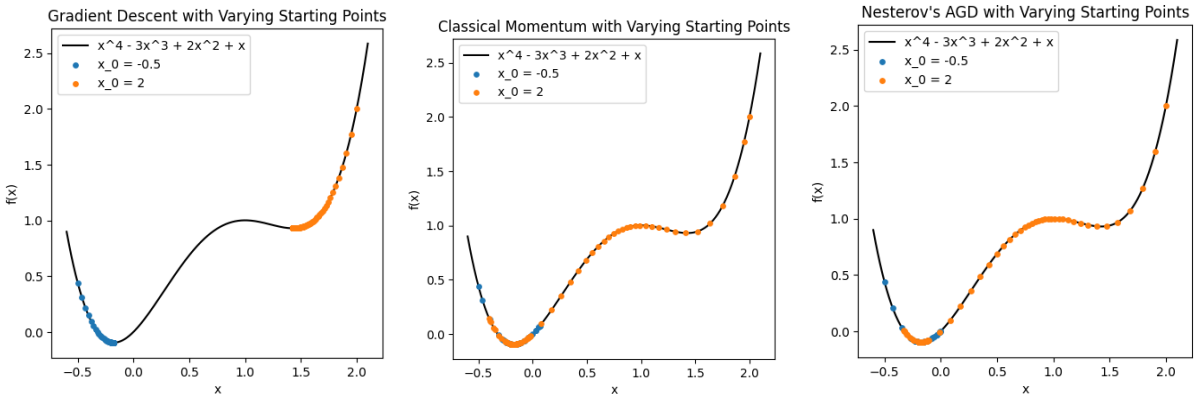

Figure 3: GD, Classical Momentum, and AGD on $f(x) = x^4 - 3x^3 + 2x^2 + x$ with $x_0 = -0.5, 2$

We can see that GD is only able to reach the global minima for the starting point $x_0 = -0.5$. For $x_0 = 2.0$, GD gets stuck at a local minimum, and isn't able to make further progress. The classical momentum and AGD algorithms, however, overcome the local minimum with their respective momentum steps, and are able to make it to the global minima with both starting points. We also note that the trajectory of classical momentum and AGD "bounce" around the global minima point once they are at the bottom-most valley due to the momentum term causing the iterates to overshoot slightly.

Algorithm	Starting Iterate	Final Iterate	Number of Iterations
GD	-0.5	-0.175	125
GD	2.0	1.425	312
Classical Momentum	-0.5	-0.175	64
Classical Momentum	2.0	-0.175	133
Nesterov's AGD	-0.5	-0.175	87
Nesterov's AGD	2.0	-0.175	116

Table 1: GD, Classical Momentum, and AGD Results

Appendix A. Python Implementation of Methods

Note that the algorithms here take a `tol` parameter, which is a termination threshold that halts the algorithm if we reach an optimal point before the max number of iterations have been reached. Also, the functions return all of the iterates as a numpy array rather than just the final iterate – this is mostly for convenience for the plots used in this paper.

Algorithm 1 Gradient Descent

```
def gradient_descent(grad_f, x0, lr=1e-3, max_iter=100, tol=1e-4):
    """
    Gradient Descent algorithm

    Parameters
    ----------
    `grad_f`: gradient function
    `x0`: initial iterate
    `lr`: learning rate, defaults to 0.001
    `max_iter`: max number of iterations to run, defaults to 100
    `tol`: termination threshold for gradient norm
    """
    xt = np.copy(x0)
    grad_f_xt = grad_f(xt)
    trajectory = [xt]

    t = 1
    while t < max_iter and np.linalg.norm(grad_f_xt) > tol:
        xt = xt - lr * grad_f_xt
        grad_f_xt = grad_f(xt)
        trajectory.append(xt)
        t += 1

    trajectory = np.array(trajectory)
    return trajectory
```

Algorithm 2 Momentum

```
def momentum(grad_f, x0, lr=1e-3, alpha=0.9, max_iter=100, tol=1e-4):
    xt = np.copy(x0)
    grad_f_xt = grad_f(xt)
    trajectory = [xt]

    t = 1
    while t < max_iter and np.linalg.norm(grad_f_xt) > tol:
        if t == 1:  # no momentum in first step
            vt = -lr * grad_f_xt
        else:
            vt = alpha * (xt - trajectory[-2]) - lr * grad_f_xt
        xt = xt + vt
        grad_f_xt = grad_f(xt)
        trajectory.append(xt)
        t += 1

    trajectory = np.array(trajectory)
    return trajectory
```

Algorithm 3 Nesterov's AGD

```python
def nesterov_agd(grad_f, x0, lr=1e-3, alpha=0.9, max_iter=100, tol=1e-4):
    xt = np.copy(x0)
    grad_f_xt = grad_f(xt)
    trajectory = [xt]

    t = 1
    while t < max_iter and np.linalg.norm(grad_f_xt) > tol:
        if t == 1:
            vt = -(1 + alpha) * lr * grad_f(xt)  # derived from Ex 5.1
        else:
            prev_xt = trajectory[-2]
            vt = alpha * (xt - prev_xt) - lr * grad_f(xt + alpha * (xt - prev_xt))
        xt = xt + vt
        grad_f_xt = grad_f(xt)
        trajectory.append(xt)
        t += 1

    trajectory = np.array(trajectory)
    return trajectory
```

Appendix B. Exercise Solutions

Solution to Exercise 4.1. This can be shown by recursively expanding the velocity terms $\mathbf{v}_i$. We have

$$\begin{aligned}
\mathbf{v}_{t+1} &= \alpha\mathbf{v}_t - \eta\nabla f(\boldsymbol{\theta}_t) \\
&= \alpha(\alpha\mathbf{v}_{t-1} - \eta\nabla f(\boldsymbol{\theta}_{t-1})) - \eta\nabla f(\boldsymbol{\theta}_t) \\
&= \alpha^2\mathbf{v}_{t-1} - \eta\alpha\nabla f(\boldsymbol{\theta}_{t-1}) - \eta\nabla f(\boldsymbol{\theta}_t) \\
&= \alpha^2(\alpha\mathbf{v}_{t-2} - \eta\nabla f(\boldsymbol{\theta}_{t-2})) - \eta\alpha\nabla f(\boldsymbol{\theta}_{t-1}) - \eta\nabla f(\boldsymbol{\theta}_t) \\
&= \alpha^3\mathbf{v}_{t-2} - \eta\alpha^2\nabla f(\boldsymbol{\theta}_{t-2}) - \eta\alpha\nabla f(\boldsymbol{\theta}_{t-1}) - \eta\nabla f(\boldsymbol{\theta}_t) \\
&\vdots \\
&= \alpha^{t+1}\mathbf{v}_0 - \eta\alpha^t\nabla f(\boldsymbol{\theta}_0) - \eta\alpha^{t-1}\nabla f(\boldsymbol{\theta}_1) - \cdots - \eta\alpha\nabla f(\boldsymbol{\theta}_{t-1}) - \eta\nabla f(\boldsymbol{\theta}_t) \\
&= \alpha^{t+1}\mathbf{v}_0 - \eta\sum_{i=0}^{t}\alpha^{t-i}\nabla f(\boldsymbol{\theta}_i). \\
&= -\eta\sum_{i=0}^{t}\alpha^{t-i}\nabla f(\boldsymbol{\theta}_i).
\end{aligned}$$

□

Solution to Exercise 5.1. Similarly to Exercise 4.1, this can be shown by recursively expanding the velocity terms $\mathbf{v}_i$. However, the expansion is a bit more involved. We have

$$\begin{aligned}
&\mathbf{v}_{t+1} \\
&= -\eta\nabla f(\boldsymbol{\theta}_t) + \alpha\mathbf{v}_t - \alpha\eta(\nabla f(\boldsymbol{\theta}_t) - \nabla f(\boldsymbol{\theta}_{t-1})) \\
&= -(1+\alpha)\eta\nabla f(\boldsymbol{\theta}_t) + \alpha\eta\nabla f(\boldsymbol{\theta}_{t-1}) + \alpha\mathbf{v}_t \\
&= -(1+\alpha)\eta\nabla f(\boldsymbol{\theta}_t) - \alpha^2\eta\nabla f(\boldsymbol{\theta}_{t-1}) + \alpha^2\eta\nabla f(\boldsymbol{\theta}_{t-2}) + \alpha^2\mathbf{v}_{t-1} \\
&= -(1+\alpha)\eta\nabla f(\boldsymbol{\theta}_t) - \alpha^2\eta\nabla f(\boldsymbol{\theta}_{t-1}) - \alpha^3\eta\nabla f(\boldsymbol{\theta}_{t-2}) + \alpha^3\eta\nabla f(\boldsymbol{\theta}_{t-3}) + \alpha^2\mathbf{v}_{t-2} \\
&\vdots \\
&= -(1+\alpha)\eta\nabla f(\boldsymbol{\theta}_t) - \alpha^2\eta\nabla f(\boldsymbol{\theta}_{t-1}) - \alpha^3\eta\nabla f(\boldsymbol{\theta}_{t-2}) - \cdots - \alpha^t\eta\nabla f(\boldsymbol{\theta}_1) + \alpha^t\eta\nabla f(\boldsymbol{\theta}_0) + \alpha^t\mathbf{v}_1
\end{aligned}$$

Now, using the derivation for $\mathbf{v}_1$ given in the exercise, we have

$$\alpha^t\eta\nabla f(\boldsymbol{\theta}_0) + \alpha^t\mathbf{v}_1 = \alpha^t\eta\nabla f(\boldsymbol{\theta}_0) - \alpha^t\eta(1+\alpha)\nabla f(\boldsymbol{\theta}_0) = -\alpha^{t+1}\eta\nabla f(\boldsymbol{\theta}_0),$$

so

$$\begin{aligned}
\mathbf{v}_{t+1} &= -(1+\alpha)\eta\nabla f(\boldsymbol{\theta}_t) - \alpha^2\eta\nabla f(\boldsymbol{\theta}_{t-1}) - \alpha^3\eta\nabla f(\boldsymbol{\theta}_{t-2}) - \cdots - \alpha^t\eta\nabla f(\boldsymbol{\theta}_1) - \alpha^{t+1}\eta\nabla f(\boldsymbol{\theta}_0) \\
&= -(1+\alpha)\eta\nabla f(\boldsymbol{\theta}_t) - \eta\sum_{i=0}^{t-1}\alpha^{t+1-i}\nabla f(\boldsymbol{\theta}_i) \\
&= -\eta\nabla f(\boldsymbol{\theta}_t) - \eta\sum_{i=0}^{t}\alpha^{t+1-i}\nabla f(\boldsymbol{\theta}_i),
\end{aligned}$$

as required. □

References

Baughman, D. and Y. Liu (1995). "2 - Fundamental and Practical Aspects of Neural Computing". In: *Neural Networks in Bioprocessing and Chemical Engineering*. Ed. by D. Baughman and Y. Liu. Boston: Academic Press, pp. 21–109. ISBN: 978-0-12-083030-5. DOI: https://doi.org/10.1016/B978-0-12-083030-5.50008-4. URL: https://www.sciencedirect.com/science/article/pii/B9780120830305500084.

Bengio, Y., N. Boulanger-Lewandowski, and R. Pascanu (2013). "Advances in optimizing recurrent networks". In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 8624–8628. DOI: 10.1109/ICASSP.2013.6639349.

Brownlee, Jason (2021). *Gradient Descent With Momentum from Scratch*. https://machinelearningmastery.com/gradient-descent-with-momentum-from-scratch/. [Online; accessed 29-April-2024].

Melville, J. (2016). *Nesterov Accelerated Gradient and Momentum*. https://jlmelville.github.io/mize/nesterov.html.

Nesterov, Y. (2013). "Gradient methods for minimizing composite functions". In: *Mathematical programming* 140.1, pp. 125–161.

Ruder, S. (2016). "An overview of gradient descent optimization algorithms". In: *CoRR* abs/1609.04747. arXiv: 1609.04747. URL: http://arxiv.org/abs/1609.04747.

Sutskever, I. et al. (2013). "On the importance of initialization and momentum in deep learning". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by S. Dasgupta and D. McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, pp. 1139–1147. URL: https://proceedings.mlr.press/v28/sutskever13.html.