Nesterov Accelerated Gradient and Momentum

December 31, 2016

December 16 2021: I have rewritten a large chunk of this document to make the derivations a little clearer and added a brief section on the definition of Nesterov momentum used in the NAdam paper.

A way to express Nesterov Accelerated Gradient (NAG) in terms of a regular momentum update was noted by Sutskever and co-workers (http://www.jmlr.org/proceedings/papers/v28/sutskever13.html), and perhaps more importantly, when it came to training neural networks, it seemed to work better than classical momentum schemes. This was further confirmed by Bengio and co-workers (https://arxiv.org/abs/1212.0901), who provided an alternative formulation that might be easier to integrate into existing software.

I implemented NAG as part of mize (https://github.com/jlmelville/mize), but found it was a bit more difficult than I'd anticipated to test that I'd got the correct results.

It seems I am not alone in being confused by the exact details of implementing NAG as a momentum scheme (e.g. comments and issues in the deep learning projects keras (https://github.com/fchollet/keras/issues/966) and mocha.jl (https://github.com/pluskid/Mocha.jl/pull/47), which I found from random googling), so mainly for the benefit of future me, I am going to derive the Sutskever and Bengio formulations from NAG in tedious detail. I am also going to derive an alternative expression that I ended up using in mize. Finally, I will write some R code to demonstrate their equivalence. This will turn out to be trickier than I anticipated.

I originally tried to stick with the notation used by Sutskever but in retrospect that was a mistake. It's too hard to keep everything straight between the different papers. I will borrow some symbols and try and be very clear about what they mean and what each expression applies to. Translating between this document and the original papers may require mentally transposing some symbols and shifting iteration indices from t to $t + 1$ or $t - 1$.

Apart from the Sutskever and Bengio papers linked to above, it's worth checking out the appendix to the Sutskever paper (PDF) (http://www.jmlr.org/proceedings/papers/v28/sutskever13-supp.pdf) or the relevant part (chapter 7) of Sutskever's thesis (http://hdl.handle.net/1807/36012). For an introduction to NAG itself, try the first part of this paper by O'Donoghue and Candès (https://arxiv.org/abs/1204.3982) (the rest of it is also good but not germane to Nesterov momentum expressions).

Definitions

The goal is to optimize some parameters, θ. This is going to involve gradient descent, so we will be evaluating the gradient of an objective function of those parameters, $\nabla f(\theta)$, and moving a certain distance in the direction of the negative of the gradient, the distance being related to the learning rate, ε. There will also be a momentum term involved, with momentum coefficient μ. The value of the parameters, learning rate and momentum at iteration t will be indicated by a subscript, e.g. θ_t.

The Update Vector

A definition which holds across all the methods discussed here is that the parameters at iteration $t+1$ are related to the parameters at iteration t by an update that involves the addition of a velocity vector, v:

$$\theta_{t+1} = \theta_t + v_{t+1} \implies v_{t+1} = \theta_{t+1} - \theta_t$$

This is pretty obvious, but the expressions we will be dealing with here use a recursive definition of v_t, i.e. to usefully define these updates we need to express v_t in terms of v_{t-1}. The exact expression depends on whether we are using classical momentum or NAG. In both cases, it would make sense to label a particular vector as v_t, but due to the similarity in how they work, the temptation to substitute the classical momentum recursive definition of v_t into an expression for NAG is almost overwhelming, and also always wrong.

To confuse matters further, some papers (see the Bengio paper for example) directly use v_t in expressions that update parameters, but where the v_t doesn't actually apply to the parameter being updated.

Finally, I'll try to be strict about names. When I'm referring to the original Nesterov Accelerated Gradient, I'll call that NAG. When I'm referring to the versions that recast it as a momentum scheme, I'll call that Nesterov momentum, and refer to either the Sutskever or Bengio forumulation where necessary. The traditional momentum scheme is referred to as "standard" or "regular" momentum by Bengio, and "classical" by Sutskever. Other authors call it "Polyak" or "heavy ball" momentum. I'll refer to it as "classical" momentum.

Classical Momentum

The classical momentum velocity vector is defined by Bengio as:

$$v_t = \mu_{t-1} v_{t-1} - \varepsilon_{t-1} \nabla f\left(\theta_{t-1}\right)$$

Sutskever gives the same definition but without any t subscript on the velocity vector or the momentum coefficient.

We can write out the full update for classical momentum as:

$$\theta_{t+1} = \theta_t + \mu_t v_t - \varepsilon_t \nabla f\left(\theta_t\right)$$

with velocity:

$$v_{t+1} = \mu_t v_t - \varepsilon_t \nabla f\left(\theta_t\right)$$

Later we will want to use the recursive definition of momentum, but it won't be the full update, so we will not be able to define it as v_t. In those circumstances, I will call the update m_t to avoid confusion:

$$m_{t+1} = \mu_t m_t - \varepsilon_t \nabla f\left(\theta_t\right)$$

Remember: $v_{t+1} = m_{t+1}$ *only* in the case of classical momentum.

NAG

The Nesterov Accelerated Gradient method consists of a gradient descent step, followed by something that looks a lot like a momentum term, but isn't exactly the same as that found in classical momentum. I'll call it a "momentum stage" here. If you look at the Sutskever paper, t's important to note that the parameters being minimized by NAG are given the symbol y, not θ. You'll see that θ is the symbol for the parameters after they've been updated by the gradient descent stage, but before the momentum stage.

I'm going to stick with θ_t as the parameters being updated. But we do need to define a symbol that accounts for the state of the parameters after the gradient descent stage. I'll go with ϕ_{t+1}. Whether you consider the intermediate step to be part of iterate t or $t+1$ doesn't really matter, but different papers use different conventions, so you will need to keep track of it all or risk getting very confused. In this case I am going with $t+1$ so it's consistent with any new quantities calculated in an iteration (like v_{t+1}).

Here's the gradient descent stage:

$$\phi_{t+1} = \theta_t - \varepsilon_t \nabla f\left(\theta_t\right)$$

followed by the momentum-like stage:

$$\theta_{t+1} = \phi_{t+1} + \mu_t\left(\phi_{t+1} - \phi_t\right)$$

That concludes one iteration of NAG. The hard part is actually finding the correct learning rate and momentum value in order to get the convergence guarantees that make the method attractive, but that needn't concern us. Sutskever in his thesis suggests manual tuning to get an optimal result (at least for deep learning applications) so we are in heuristic territory here anyway.

Sutskever Nesterov Momentum

The key idea behind the Sutskever momentum derivation is to shift the perspective about which of the parameters we want as the result of the iteration, from θ to ϕ. Rather than having the optimization iterations proceed as "gradient descent, momentum (end of iteration 1), gradient descent, momentum (end of iteration 2), gradient descent etc." move the boundary of where the iterations end by a half-iteration to get "momentum, gradient descent (end of iteration 1), momentum, gradient descent (end of iteration 2) etc.". This leaves a phantom gradient descent step that used to be first stage of the first iteration, now floating in the nether regions of iteration zero, but you can just pretend that the starting position is the result of gradient descent from some other arbitrary starting position.

Perhaps a visualization will help. Here are two full iterations of NAG in its standard form, separated by boxes so you can see where one interation ends and the other begins.

$$\boxed{\begin{aligned}\phi_{t+1} &= \theta_t - \varepsilon_t \nabla f\left(\theta_t\right)\\ \theta_{t+1} &= \phi_{t+1} + \mu_t\left(\phi_{t+1} - \phi_t\right)\end{aligned}}$$

$$\boxed{\begin{aligned}\phi_{t+2} &= \theta_{t+1} - \varepsilon_{t+1} \nabla f\left(\theta_{t+1}\right)\\ \theta_{t+2} &= \phi_{t+2} + \mu_{t+1}\left(\phi_{t+2} - \phi_{t+1}\right)\end{aligned}}$$

The Sutskever derivation says shift the boundaries of the iteration so that it starts one stage later. This is the same four equations as above, but with a box showing the new definition of the iteration:

$$\phi_{t+1} = \theta_t - \varepsilon_t \nabla f\left(\theta_t\right)$$

$$\boxed{\begin{aligned}\theta_{t+1} &= \phi_{t+1} + \mu_t\left(\phi_{t+1} - \phi_t\right) \\ \phi_{t+2} &= \theta_{t+1} - \varepsilon_{t+1}\nabla f\left(\theta_{t+1}\right)\end{aligned}}$$

$$\theta_{t+2} = \phi_{t+2} + \mu_{t+1}\left(\phi_{t+2} - \phi_{+1}\right)$$

Now that the definition of where we start and finish each iteration has changed we need to re-label some variables: uses of ϕ and ε need to be shifted back one step of t to give:

$$\begin{aligned}\theta_{t+1} &= \phi_t + \mu_t\left(\phi_t - \phi_{t-1}\right) \\ \phi_{t+1} &= \theta_{t+1} - \varepsilon_t \nabla f\left(\theta_{t+1}\right)\end{aligned}$$

Then because our update is now in terms of ϕ rather than θ, we can label $\phi_t - \phi_{t-1}$ as v_t. The new definition of θ_{t+1} is then:

$$\theta_{t+1} = \phi_t + \mu_t v_t$$

And we can now substitute that definition for θ_{t+1} into the momentum stage expression, which lets us write this half-shifted NAG iteration in one line:

$$\phi_{t+1} = \phi_t + \mu_t v_t - \varepsilon_t \nabla f\left(\phi_t + \mu_t v_t\right)$$

There you have it: this looks just like the classical momentum update, except that the gradient is calculated after the momentum update. Hence, one can do NAG by simply reversing the order in which the update is usually carried out: do the momentum stage first, update the parameters, and then do the gradient descent part. Just like Sutskever said you could.

Bengio Nesterov Momentum

I found working through the Bengio version of Nesterov momentum a bit brain-breaking. It starts from the Sutskever definition and then defines a new variable, Θ, which represents the parameters after the momentum update. But that just moves us one half-step forward in the iteration, so we are back to doing things in the original NAG way: gradient descent then momentum-style update. So their paper's Θ is the same as our θ. Its explanation of "committing to the 'peekedahead' parameters" and then "backtracking by the same amount before each update", also never landed with me (this is a reflection on my mental capacity, not the paper). Finally, it expresses the update in terms of v, but that's still the the update in terms of the difference in ϕ, not θ. Unlike this document it also uses $t-1$ and t to index the parameters that are being updated, so you will have to remember to add or subtract 1 from the subscripts when looking between here and the paper, but that's the least of your problems honestly.

It's all too much. I can't get their explanation to stick in my brain, even though I can follow the individual steps and get the right expression. So I will show you how to arrange the standard NAG expressions into their form, rather than starting from the Sutskever result.

First, let's remove the confusion around using v as an update vector which isn't the change in the parameters that we are updating each iteration. Instead of $\phi_{t+1} - \phi_t = v_{t+1}$, define:

$$\phi_{t+1} - \phi_t = b_{t+1}$$

(the 'b' is for Bengio). Our goal now is to end up with an expression that is defined using θ_t (no ϕ_t allowed) and b_t.

Now we are going to go back to using the original NAG expression so to help you scrub all the shifts in t from your mind here is the expression again:

$$\begin{aligned}\phi_{t+1} &= \theta_t - \varepsilon_t \nabla f\left(\theta_t\right) \\ \theta_{t+1} &= \phi_{t+1} + \mu_t\left(\phi_{t+1} - \phi_t\right)\end{aligned}$$

Start by replacing all uses of ϕ_{t+1} in the second equation with the definition in terms of θ_t given in the first equation:

$$\begin{aligned}\theta_{t+1} &= \theta_t - \varepsilon_t \nabla f\left(\theta_t\right) + \mu_t\left(\theta_t - \varepsilon_t \nabla f\left(\theta_t\right) - \phi_t\right) \\ &= \left(1 + \mu\right)\theta_t - \left(1 + \mu_t\right)\varepsilon_t \nabla f\left(\theta_t\right) - \mu_t \phi_t\end{aligned}$$

So far, so un-promising. Time to get b_t involved. Remember how we defined:

$$\begin{aligned}\theta_{t+1} &= \phi_{t+1} + \mu_t\left(\phi_{t+1} - \phi_t\right) \\ &= \phi_{t+1} + \mu_t b_{t+1}\end{aligned}$$

Let's shift everything back one iteration:

$$\begin{aligned}\theta_t &= \phi_t + \mu_{t-1} b_t \\ \implies \phi_t &= \theta_t - \mu_{t-1} b_t\end{aligned}$$

And now substitute for ϕ_t in the last term in our NAG update:

$$\begin{aligned}\theta_{t+1} &= \left(1 + \mu\right)\theta_t - \left(1 + \mu_t\right)\varepsilon_t \nabla f\left(\theta_t\right) - \mu_t \phi_t \\ &= \left(1 + \mu\right)\theta_t - \left(1 + \mu_t\right)\varepsilon_t \nabla f\left(\theta_t\right) - \mu_t \theta_t + \mu_{t-1}\mu_t b_t\end{aligned}$$

The final step is to gather together the θ_t terms and the result is:

$$\theta_{t+1} = \theta_t + \mu_{t-1}\mu_t b_t - \left(1 + \mu_t\right)\varepsilon_t \nabla f\left(\theta_t\right)$$

Ta da. There it is, just like in the paper, subject to the differences in symbols and when the iteration ticks over from t to $t + 1$ and so on.

The advantage of this expression for the Nesterov momentum is that it doesn't require calculating a gradient at a non-standard position, and only requires a modification to the coefficients used to calculate the velocity, which is probably an easier change to make to an existing codebase which already uses classical momentum. However, because this expression is recursive on b_t rather than the update vector v_t, you will need to remember to store the a different vector than you would with classical momentum. It's not really clear to me whether that is an easy or hard thing to do with most optimization code bases vs having to make code changes to calculate the gradient at a different location than the current parameter vector, as you have to do with the Sutskever version.

You should also remember that while the parameters from this version and the Sutskever version should converge to the same values from the same starting positions, on an iteration-to-iteration version level, you will get different values for the parameters, because they are half a step out of sync with each other.

An Alternative Expression for NAG

Let's go back to the original formulation of NAG and now instead of making θ the variables after gradient descent, we'll put them where y was originally used. The variables after gradient descent I'll refer to as ϕ.

$$\phi_t = \theta_t - \varepsilon_t \nabla f(\theta_t)$$

$$\theta_{t+1} = \phi_t + \mu_t (\phi_t - \phi_{t-1})$$

Now, let's just write out the momentum stage in terms of θ, substituting ϕ wherever we find it:

$$\theta_{t+1} = \theta_t - \varepsilon_t \nabla f(\theta_t) + \mu_t [\theta_t - \varepsilon_t \nabla f(\theta_t) - \theta_{t-1} + \varepsilon_{t-1} \nabla f(\theta_{t-1})]$$

Rearranging:

$$\theta_{t+1} = \theta_t + \mu_t [\theta_t - \theta_{t-1} + \varepsilon_{t-1} \nabla f(\theta_{t-1}) - \varepsilon_t \nabla f(\theta_t)] - \varepsilon_t \nabla f(\theta_t)$$

Finally, we can subtitute in v_t for the first two terms in the square brackets, to give:

$$\theta_{t+1} = \theta_t + \mu_t [v_t + \varepsilon_{t-1} \nabla f(\theta_{t-1}) - \varepsilon_t \nabla f(\theta_t)] - \varepsilon_t \nabla f(\theta_t)$$

with velocity:

$$v_{t+1} = \mu_t [v_t + \varepsilon_{t-1} \nabla f(\theta_{t-1}) - \varepsilon_t \nabla f(\theta_t)] - \varepsilon_t \nabla f(\theta_t)$$

This looks a lot like the classical momentum expression, but with the velocity vector modified to first remove the contribution of the gradient descent from the previous iteration, and replace it with the gradient descent contribution from the *current* iteration. Gives an interesting insight into the idea of the Nesterov momentum using a form of "lookahead" with the gradient descent.

You could also choose to expand the velocity expression to make it look a bit like the Bengio formulation:

$$v_{t+1} = \mu_t [v_t + \varepsilon_{t-1} \nabla f(\theta_{t-1})] - (1 + \mu_t) \varepsilon_t \nabla f(\theta_t)$$

but as this version can't be expressed as the classical momentum form with different coefficients, the way the Bengio formulation can be, it probably doesn't gain you anything in terms of implementation, except you can expand it and rearrange it further to give:

$$v_{t+1} = \mu_t v_t - \varepsilon_t \nabla f(\theta_t) + \mu_t [\varepsilon_{t-1} \nabla f(\theta_{t-1}) - \varepsilon_t \nabla f(\theta_t)]$$

which now resembles the classical momentum expression with an extra momentum term. I haven't found a definitive reference for this expression, or what, if any, extra insight it provides, but user 'denis' uses this expression in an answer on the Cross Validated Stack Exchange (https://stats.stackexchange.com/a/233430), and refers to the third form as "gradient momentum". Another way to look at this is to say that the third term reduces the contribution of the classical momentum component of the update and increases the contribution of the gradient descent, with the degree of weighting controlled by μ_t.

Quasi-hyperbolic momentum (https://arxiv.org/abs/1810.06801) uses a similar formulation, but introduces an extra parameter that allows QHM to generalize over classical momentum and NAG, subject to the restriction of a fixed learning rate. The Stack Exchange answer given above also suggests that the "gradient

momentum" term can be set independently of μ and Denis Yarats is one of the authors of the QHM paper, so perhaps we see the origins of QHM in this answer.

The setup and presentation of the update equations for QHM in that paper make it a bit hard to see the connection between it and the form of NAG given here. The supplemental material in Understanding the Role of Momentum in Stochastic Gradient Methods (https://papers.nips.cc/paper/9158-understanding-the-role-of-momentum-in-stochastic-gradient-methods) might be of interest, or some extra notes on QHM (https://jlmelville.github.io/mize/qhm.html) I wrote for my own benefit.

Unified Momentum

December 12 2021: Previously in this section I wrote down a QHM-like expression I called "generalized" momentum. This was a bad choice of words because there is already a concept in physics called generalized momentum, which is nothing to do with this. Fortunately, in 2018 Zou and co-workers (https://arxiv.org/abs/1808.03408) published a paper with the same expression and called it "unified stochastic momentum". The stochastic bit is because it is in the context of stochastic gradient descent but that has no effect on the expression. So let's call it "unified momentum" instead:

Similarly to QHM, we can write a related ~~"generalized"~~ "unified" momentum update by introducing a parameter β to give:

$$v_{t+1} = \mu_t v_t - \varepsilon_t \nabla f\left(\theta_t\right) + \beta_t \mu_t \left[\varepsilon_{t-1} \nabla f\left(\theta_{t-1}\right) - \varepsilon_t \nabla f\left(\theta_t\right)\right]$$

where setting $\beta_t = 0$ will give classical momentum and $\beta_t = 1$ gives Nesterov.

In 2020, a similar expression was given by Ziyin and co-workers (https://arxiv.org/abs/2002.04839), but didn't give it a catchy name. They don't cite the Zou paper, so presumably they discovered it independently. I'd like to find the earliest use of this interpolation, but neither the Zou nor the Ziyin paper give a reference to any earlier paper. Perhaps it's just incredibly obvious?

From the look of the expression, it seems that a major downside of this expression is that calculating the parameters for iteration $t + 1$ requires storage of information from not only iteration t but from iteration $t - 1$ too. But you don't necessarily have to do any extra storage with an implementation that used this version of NAG. At the end of an iteration, when saving the velocity vector for the next iteration, you change:

$$v_{t-1} \leftarrow v_t$$

to:

$$v_{t-1} \leftarrow v_t + \varepsilon_t \nabla f\left(\theta_t\right)$$

and then when calculating the momentum term, change:

$$\mu_t v_t$$

to:

$$\mu_t \left[v_t - \varepsilon_t \nabla f\left(\theta_t\right)\right]$$

Unrolling NAG and Classical Momentum

As a way to understand the difference between classical momentum and NAG, let's write out the first few steps of the optimization, substituting in the recursive definitions with the previous step, and see what emerges.

Because I have finally got tired of writing out $\varepsilon_t \nabla f\left(\theta_t\right)$ all over the place (and it's visually distracting), let's define:

$$-\varepsilon_t \nabla f\left(\theta_t\right) \equiv s_t$$

to represent the steepest descent direction. Note that this includes the negative sign. Also, I am going to assume that the momentum is constant, to save on some more notational clutter.

Initial coordinates are at θ_0.

Classical Momentum

Using the slightly briefer symbols, the classical momentum update is:

$$\theta_{t+1} = \theta_t + s_t + \mu v_t$$

And the first four updates (including steepest descent on the first iteration) are as follows:

$$\begin{aligned}
\theta_1 &= \theta_0 + s_0 \\
\\
\theta_2 &= \theta_1 + s_1 + \mu v_1 \\
&= \theta_1 + s_1 + \mu s_0 \\
\\
\theta_3 &= \theta_2 + s_2 + \mu v_2 \\
&= \theta_2 + s_2 + \mu\left(s_1 + \mu s_0\right) \\
&= \theta_2 + s_2 + \mu s_1 + \mu^2 s_0 \\
\\
\theta_4 &= \theta_3 + s_3 + \mu v_3 \\
&= \theta_3 + s_3 + \mu\left(s_2 + \mu s_1 + \mu^2 s_0\right) \\
&= \theta_3 + s_3 + \mu s_2 + \mu^2 s_1 + \mu^3 s_0
\end{aligned}$$

NAG

We'll use this straight-forward expression for NAG:

$$\theta_{t+1} = \theta_t + s_t + \mu\left(\theta_t + s_t - \theta_{t-1} - s_{t-1}\right)$$

with just this bit of re-arranging so terms are all grouped together and the v_t is at the end, which makes it easier to follow the substitution of v_t:

$$\theta_{t+1} = \theta_t + \left(1 + \mu\right) s_t - \mu s_{t-1} + \mu v_t$$

The first step

December 14 2021 (this section describing your choices for the first step is new).

Like classical momentum, the only sensible direction to move in on the first step is the steepest descent direction because we have no "memory" of any previous directions to use. But a strict application of the NAG update would set $v_0 = 0$ and $s_{-1} = 0$ and leave us with:

$$\theta_1 = \theta_0 + (1 + \mu)\, s_0$$

This is still moving in the direction of the negative gradient, but it takes a longer step size than classical momentum. This doesn't make a lot of sense to me as a good first step without any contributions from previous updates or gradients. Why have a momentum term involved at all when most of what that μ gets applied to in subsequent iterations isn't around?

Also, as we will see later, taking the long step makes reconciling the different versions of NAG a bit harder than it needs to be . Plus, I see the appeal of both classical momentum and NAG taking the step on the first iteration when there is no history of updates to work on. So I won't blame you if you decide to go with steepest descent with step size of length ε_1 rather than $(1 + \mu)\, \varepsilon_1$.

Unfortunately, as you will discover, your efforts to avoid taking the "long" steepest descent step will be in vain. If you don't do it on the first iteration, it just shows up again in the second iteration.

We'll look at the results with both choices, and it's honestly not that consequential, but probably going with the $(1 + \mu)$ step size on the first iteration is the right move.

Having decided on the "long" version of the first steepest descent step, here it is along with the subsequent three steps of NAG:

$$\begin{aligned}
\theta_1 &= \theta_0 + (1 + \mu)\, s_0 \\
\\
\theta_2 &= \theta_1 + (1 + \mu)\, s_1 - \mu s_0 + \mu v_1 \\
&= \theta_1 + (1 + \mu)\, s_1 - \mu s_0 + \mu\,(1 + \mu)\, s_0 \\
&= \theta_1 + (1 + \mu)\, s_1 + \mu^2 s_0 \\
\\
\theta_3 &= \theta_2 + (1 + \mu)\, s_2 - \mu s_1 + \mu v_2 \\
&= \theta_2 + (1 + \mu)\, s_2 - \mu s_1 + \mu\,(1 + \mu)\, s_1 + \mu^3 s_0 \\
&= \theta_2 + (1 + \mu)\, s_2 + \mu^2 s_1 + \mu^3 s_0 \\
\\
\theta_4 &= \theta_3 + (1 + \mu)\, s_3 - \mu s_2 + \mu v_3 \\
&= \theta_3 + (1 + \mu)\, s_3 - \mu s_2 + \mu\left[(1 + \mu)\, s_2 + \mu^2 s_1 + \mu^3 s_0\right] \\
&= \theta_3 + (1 + \mu)\, s_3 - \mu s_2 + \mu\,(1 + \mu)\, s_2 + \mu^3 s_1 + \mu^4 s_0 \\
&= \theta_3 + (1 + \mu)\, s_3 + \mu^2 s_2 + \mu^3 s_1 + \mu^4 s_0
\end{aligned}$$

You can see a pattern emerging here at this point, I hope. These are quite similar in form compared to the classical momentum.

NAG with "short" steepest descent

What happens if we decide to go with the shorter steepest descent step and make the same first step as we do with classical momentum? Here's how that plays out:

$$\theta_1 = \theta_0 + s_0$$

$$\begin{aligned}\theta_2 &= \theta_1 + (1+\mu)\, s_1 - \mu s_0 + \mu v_1 \\ &= \theta_1 + (1+\mu)\, s_1 - \mu s_0 + \mu s_0 \\ &= \theta_1 + (1+\mu)\, s_1\end{aligned}$$

$$\begin{aligned}\theta_3 &= \theta_2 + (1+\mu)\, s_2 - \mu s_1 + \mu v_2 \\ &= \theta_2 + (1+\mu)\, s_2 - \mu s_1 + \mu\,(1+\mu)\, s_1 \\ &= \theta_2 + (1+\mu)\, s_2 + \mu^2 s_1\end{aligned}$$

$$\begin{aligned}\theta_4 &= \theta_3 + (1+\mu)\, s_3 - \mu s_2 + \mu v_3 \\ &= \theta_3 + (1+\mu)\, s_3 - \mu s_2 + \mu\left[(1+\mu)\, s_2 + \mu^2 s_1\right] \\ &= \theta_3 + (1+\mu)\, s_3 - \mu s_2 + \mu\,(1+\mu)\, s_2 + \mu^3 s_1 \\ &= \theta_3 + (1+\mu)\, s_3 + \mu^2 s_2 + \mu^3 s_1\end{aligned}$$

Overall the results aren't that different: nearly all of the terms are present with the same coefficients. However we can see that, as I mentioned above, the second step ends up being steepest descent as well, because the contribution of the s_1 terms cancel out. And once it's gone, it's gone for good: all the subsequent iterations lack the contribution of the s_1 term. And the second iteration is the "long" steepest descent step, the thing we were trying to avoid in the first place.

So that's two disadvantages to deal with: first, the long steepest descent is going to happen anyway. Second, the disappearance of the contribution of the first gradient after the first iteration. None of the other gradients behave like that and it also slightly messes with the relationship between the weights we get with NAG and classical momentum.

On balance, I recommend using the "long" step for the first iteration of NAG. If I need to distinguish between the two methods I will call the version where we use the "short" step "NAGs".

Relative Weights

Here's a table showing how on the 4th step, the two methods weight the current and previous gradients in deciding on the next direction:

Momentum Type	s_3	s_2	s_1	s_0
Classical	1	μ	μ^2	μ^3
NAG	1 + μ	μ^2	μ^3	μ^4
NAGs	1 + μ	μ^2	μ^3	0

From this it's clear that the difference between NAG and classical momentum is that NAG puts more weight on recent gradients: another way of looking at it is to say that NAG forgets old gradients more quickly.

The regularity of the difference between classical momentum and NAG weights suggest that there should be a way to express the NAG update in terms of a classical momentum update. See the section on 'Dozat Nesterov momentum' below for more on that.

Let's write some R code to generate a table to compare how the relative weights turn out between CM and NAG at different values of μ. We'll normalize the contributions so they sum to 1.

```
cm_weights <- function(mu) { c(1, mu, mu * mu, mu * mu * mu) }

nags_weights <- function(mu) { c(1 + mu, mu * mu, mu * mu * mu, 0) }

nag_weights <- function(mu) { c(1 + mu, mu * mu, mu * mu * mu, mu * mu * mu * mu) }


mumat <- function(wfun, mus = c(0.1, 0.25, 0.5, 0.75, 0.9, 0.99)) {
  rel_weights <- matrix(nrow = length(mus), ncol = 5)
  for (i in 1:length(mus)) {
    mu <- mus[i]
    weights <- wfun(mu)
    rel_weights[i, ] <- c(mu, weights / sum(weights))
  }
  colnames(rel_weights) <- c("mu", "s3", "s2", "s1", "s0")

  rel_weights
}
```

Classical momentum weights

```
knitr::kable(mumat(cm_weights), digits = 4)
```

mu	s3	s2	s1	s0
0.10	0.9001	0.0900	0.0090	0.0009
0.25	0.7529	0.1882	0.0471	0.0118
0.50	0.5333	0.2667	0.1333	0.0667
0.75	0.3657	0.2743	0.2057	0.1543
0.90	0.2908	0.2617	0.2355	0.2120
0.99	0.2538	0.2512	0.2487	0.2462

NAG weights

```
knitr::kable(mumat(nag_weights), digits = 4)
```

mu	s3	s2	s1	s0
0.10	0.9900	0.0090	0.0009	0.0001
0.25	0.9384	0.0469	0.0117	0.0029
0.50	0.7742	0.1290	0.0645	0.0323
0.75	0.5736	0.1844	0.1383	0.1037
0.90	0.4640	0.1978	0.1780	0.1602
0.99	0.4060	0.2000	0.1980	0.1960

NAGs weights

```
knitr::kable(mumat(nags_weights), digits = 4)
```

mu	s3	s2	s1	s0
0.10	0.9901	0.0090	0.0009	0
0.25	0.9412	0.0471	0.0118	0
0.50	0.8000	0.1333	0.0667	0
0.75	0.6400	0.2057	0.1543	0
0.90	0.5525	0.2355	0.2120	0
0.99	0.5050	0.2487	0.2462	0

At high momentum, classical momentum is only placing around 25-30% of the weight on the current gradient, whereas for NAG, it's 45-50%. For NAGs, the weight is even higher at 50-55%.

Unrolling with a momentum schedule

I kept μ constant in the unrolling expressions so far to keep things neat and to highlight some similarities between CM and NAG. But what if we want μ to vary with iteration? In our derivations of Nesterov momentum schemes we have explicitly accounted for μ_t so let's take a look (and brace ourselves for the expressions to lose a lot of readability):

Classical Momentum with a Schedule

$$\theta_1 = \theta_0 + s_0$$

$$\begin{aligned}\theta_2 &= \theta_1 + s_1 + \mu_1 v_1 \\ &= \theta_1 + s_1 + \mu_1 s_0\end{aligned}$$

$$\begin{aligned}\theta_3 &= \theta_2 + s_2 + \mu_2 v_2 \\ &= \theta_2 + s_2 + \mu_2 \left(s_1 + \mu_1 s_0\right) \\ &= \theta_2 + s_2 + \mu_2 s_1 + \mu_2 \mu_1 s_0\end{aligned}$$

$$\begin{aligned}\theta_4 &= \theta_3 + s_3 + \mu_3 v_3 \\ &= \theta_3 + s_3 + \mu_3 \left(s_2 + \mu_2 s_1 + \mu_2 \mu_1 s_0\right) \\ &= \theta_3 + s_3 + \mu_3 s_2 + \mu_3 \mu_2 s_1 + \mu_3 \mu_2 \mu_1 s_0\end{aligned}$$

Already this is looking pretty gross.

NAG with a Momentum Schedule

$$\theta_1 = \theta_0 + \left(1 + \mu_0\right) s_0$$

$$\begin{aligned}\theta_2 &= \theta_1 + \left(1 + \mu_1\right) s_1 - \mu_1 s_0 + \mu_1 v_1 \\ &= \theta_1 + \left(1 + \mu_1\right) s_1 - \mu_1 s_0 + \mu_1 \left(1 + \mu_0\right) s_0 \\ &= \theta_1 + \left(1 + \mu_1\right) s_1 + \mu_1 \mu_0 s_0\end{aligned}$$

$$\begin{aligned}\theta_3 &= \theta_2 + \left(1 + \mu_2\right) s_2 - \mu_2 s_1 + \mu_2 v_2 \\ &= \theta_2 + \left(1 + \mu_2\right) s_2 - \mu_2 s_1 + \mu_2 \left(1 + \mu_1\right) s_1 + \mu_2 \mu_1 \mu_0 s_0 \\ &= \theta_2 + \left(1 + \mu_2\right) s_2 + \mu_2 \mu_1 s_1 + \mu_2 \mu_1 \mu_0 s_0\end{aligned}$$

$$\begin{aligned}\theta_4 &= \theta_3 + \left(1 + \mu_3\right) s_3 - \mu_3 s_2 + \mu_3 v_3 \\ &= \theta_3 + \left(1 + \mu_3\right) s_3 - \mu_3 s_2 + \mu_3 \left[\left(1 + \mu_2\right) s_2 + \mu_2 \mu_1 s_1 + \mu_2 \mu_1 \mu_0 s_0\right] \\ &= \theta_3 + \left(1 + \mu_3\right) s_3 - \mu_3 s_2 + \mu_3 \left(1 + \mu_2\right) s_2 + \mu_3 \mu_2 \mu_1 s_1 + \mu_3 \mu_2 \mu_1 \mu_0 s_0 \\ &= \theta_3 + \left(1 + \mu_3\right) s_3 + \mu_3 \mu_2 s_2 + \mu_3 \mu_2 \mu_1 s_1 + \mu_3 \mu_2 \mu_1 \mu_0 s_0\end{aligned}$$

I can see why I was in no hurry to write all that out in full until now. No point showing the "NAGs" version for this, it's the same as with the constant μ case: the s_0 term vanishes after the first iteration.

Here's a comparison of the weights applied to the different gradients at θ_4:

Momentum Type	s_3	s_2	s_1	s_0
Classical	1	μ_3	$\mu_3 \mu_2$	$\mu_3 \mu_2 \mu_1$
NAG	1 + μ_3	$\mu_3 \mu_2$	$\mu_3 \mu_2 \mu_1$	$\mu_3 \mu_2 \mu_1 \mu_0$

Now the momentum is not constant at each iteration, we see a more complex relationship between the CM and NAG weights. There definitely doesn't seem to be a simple re-weighting that could map from CM to NAG here: each gradient needs re-weighting by a different μ_t.

Dozat Nesterov Momentum

18 December 2021: A new section (currently in a serviceable but somewhat unsatisfactory state) on the expression for Nesterov momentum given by Timothy Dozat in the NAdam paper.

A paper by Dozat integrating Nesterov momentum (https://openreview.net/forum?id=OM0jvwB8jIp57ZJjtNEZ) into the Adam (https://arxiv.org/abs/1412.6980) stochastic gradient descent method shows how to get from the CM to NAG by re-weighting the CM update. As we just saw, allowing μ to vary at each iterate has complicated the relationship between CM and NAG, so we need to define a new CM-like update.

Back when I defined CM, I said it would be useful to define the recursive momentum equation as m_t, identifying as separate from the parameter update, v_t, even though they coincide in CM. Here's where it comes in useful to have separated those two concepts. We now need a CM-like definition, which differs from m_t only in that we use the momentum coefficient from the previous iterate. This new definition I will label as m'_t:

$$
\begin{aligned}
m_{t+1} &= \mu_t m_t + s_t \\
m'_{t+1} &= \mu_{t-1} m'_t + s_t
\end{aligned}
$$

For NAG, the update vector v_{t+1} can now be defined in terms of m'_{t+1}, using the current iterate's momentum coefficient:

$$
v_{t+1} = \mu_t m'_{t+1} + s_t
$$

This is like doing momentum twice per iteration (I'd love to say "double momentum" but that's already a thing (https://arxiv.org/abs/2102.07367)). If we expand the update to be in terms of m'_t rather than m'_{t+1} we get:

$$
\begin{aligned}
v_{t+1} &= \mu_t \left(\mu_{t-1} m'_t + s_t \right) + s_t \\
&= \mu_{t-1} \mu_t m'_t + \left(1 + \mu_t \right) s_t
\end{aligned}
$$

So here's the overall Dozat Nesterov momentum update, rearranged a bit to make the forthcoming unrolling easier to write out:

$$
\theta_{t+1} = \theta_t + \left(1 + \mu_t \right) s_t + \mu_{t-1} \mu_t m'_t
$$

At this point, I would love to say that I can get to this expression for NAG starting from the expression we have been using so far. But I admit that it's not obvious to me how to get from the general recursive-in-v_t expression to one which is recursive in m'_t. The key maye be to notice the connection between this expression and two others: the Bengio Nesterov momentum expression and the expanded and rearranged NAG expression:

$$
\begin{aligned}
\theta_{t+1} &= \theta_t + \left(1 + \mu_t \right) s_t + \mu_{t-1} \mu_t m'_t \\
\theta_{t+1} &= \theta_t + \left(1 + \mu_t \right) s_t + \mu_{t-1} \mu_t b_t \\
\theta_{t+1} &= \theta_t + \left(1 + \mu_t \right) s_t + \mu_t \left(v_t - s_{t-1} \right)
\end{aligned}
$$

This means that:

$$m'_t = b_t = \frac{(v_t - s_{t-1})}{\mu_{t-1}}$$

Perhaps one day I will return to this section and fill in the obvious details. For now, let's just unroll the first few iterations of the Dozat expression:

$$\begin{aligned}
m'_0 &= 0 \\
\theta_1 &= \theta_0 + (1 + \mu_0) s_0 + \mu_0 \mu_{-1} m'_0 \\
\theta_1 &= \theta_0 + (1 + \mu_0) s_0
\end{aligned}$$

$$\begin{aligned}
m'_1 &= \mu_{-1} m'_0 + s_0 \\
&= s_0 \\
\theta_2 &= \theta_1 + (1 + \mu_1) s_1 + \mu_1 \mu_0 m'_1 \\
&= \theta_1 + (1 + \mu_1) s_1 + \mu_1 \mu_0 s_0
\end{aligned}$$

$$\begin{aligned}
m'_2 &= \mu_0 m'_1 + s_1 \\
&= \mu_0 s_0 + s_1 \\
\theta_3 &= \theta_2 + (1 + \mu_2) s_2 + \mu_2 \mu_1 m'_2 \\
&= \theta_2 + (1 + \mu_2) s_2 + \mu_2 \mu_1 s_1 + \mu_2 \mu_1 \mu_0 s_0
\end{aligned}$$

$$\begin{aligned}
m'_3 &= \mu_1 m'_2 + s_2 \\
&= \mu_1 \mu_0 s_0 + \mu_1 s_1 + s_2 \\
\theta_4 &= \theta_3 + (1 + \mu_3) s_3 + \mu_3 \mu_2 m'_3 \\
&= \theta_3 + (1 + \mu_3) s_3 + \mu_3 \mu_2 s_2 + \mu_3 \mu_2 \mu_1 s_1 + \mu_3 \mu_2 \mu_1 \mu_0 s_0
\end{aligned}$$

Yep, that's the NAG update alright.

Note that for this implementation, you *don't* store the previous update, v_t, but the momentum update m'_t. Also, you need to store the previous iterate's momentum coefficient μ_{t-1}.

The QHM-Dozat Nesterov Momentum Connection

Quasi-hyperbolic momentum and the Dozat version of Nesterov momentum have a very similar form: QHM can be thought of as a generalization of Dozat Nesterov momentum, by replacing μ_{t-1} in the definition of m'_{t+1} with a new free parameter, ν_t. To avoid any confusion let's call this new update q_{t+1}:

$$\begin{aligned}
q_{t+1} &= \nu_t q_t + s_t \\
v_{t+1} &= \mu_t q_{t+1} + s_t \\
&= \mu_t \nu_t q_t + (1 + \mu_t) s_t
\end{aligned}$$

Setting:

- $\nu_t = 0$ and you get steepest descent (nearly).
- $\nu_t = \mu_{t-1}$ and you get Nesterov momentum.
- $\nu_t = 1$ and you get classical momentum (nearly).

So ν_t gives you an extra parameter along with μ_t to tune how much weight should go on previous gradients when constructing the update vector.

At this point I should note that this isn't *exactly* the QHM update, although it is quite close. Also, you only get "nearly" steepest decent and CM from this equation: the step size for steepest descent is the "long" version we have grown all too accustomed to. The QHM paper doesn't have the same issue mainly because it takes care to both separate the *direction* of the update vector versus its magnitude. This takes us further away from how NAG and classical momentum are often defined, however.

NAG in practice

So that's how it's all supposed to work in principle. Here I'll demonstrate the equivalence of these methods, by implementing them all in simple R code.

The biggest simplification I'll make is that I'll assume a constant learning rate and a constant momentum coefficient.

Classical Momentum

```
#' Optimization by Classical Momentum
#'
#' @param par Starting point of vector of parameters to optimize.
#' @param fn Objective function to optimize. Takes vector with length of
#' \code{par} and returns a scalar.
#' @param gr Gradient of the objective function \code{fn}. Takes vector with
#' length of \code{par} and returns a vector of the same length.
#' @param lr Learning rate.
#' @param mu Momentum coefficient.
#' @param max_iter Maximum number of iterations to optimize for. First iteration
#' is always steepest descent.
#' @return list with components: \code{par} final set of parameters; \code{f}
#' value of \code{fn} evaluated at the returned set of parameters; \code{fs}
#' vector of function evaluated after each iteration.
cm <- function(par, fn, gr, lr, mu, max_iter = 10) {
  fs <- rep(0, max_iter)

  v <- rep(0, length(par))
  for (i in 1:max_iter) {
    g <- gr(par)
    v <- mu * v - lr * g
    par <- par + v

    # store results
    f <- fn(par)
    fs[i] <- f
  }

  list(par = par, f = f, fs = fs)
}
```

This is a reference implementation of classical momentum. Nearly all the parameters and the return values are the same for the other functions (except where noted), so they're documented here once.

Onto the various NAG implementations. They all have an extra bit of code in them to ensure that on the first iteration only the gradient descent stage is carried out. This is needed to keep the outputs consistent between the different implementations. The `cm` routine doesn't require any extra checks for steepest descent on the first iteration, because initializing the velocity vector to zero results in steepest descent anyway even if the momentum coefficient is non-zero. As we'll see below, some of the other implementations either don't explicitly use a velocity vector or need the momentum coefficient to be set to zero on the first iteration to get the same result.

NAG

```
# Optimization by Nesterov Accelerated Gradient
#
# Return list also contains gd_fs: function evaluated after gradient descent
# stage of each iteration; all: function evaluated after gradient descent
# stage and momentum stage, in order.
nag <- function(par, fn, gr, lr, mu, max_iter = 10) {
  fs <- rep(0, max_iter)
  gd_fs <- rep(0, max_iter)
  all <- rep(0, max_iter * 2)

  x_old <- rep(0, length(par))
  for (i in 1:max_iter) {
    # gradient descent stage
    g <- gr(par)
    x <- par - (lr * g)

    # store gradient descent values
    f <- fn(x)
    gd_fs[i] <- f
    all[i * 2 - 1] <- f

    # momentum stage and update
    par <- x + ifelse(i == 1, 0, mu) * (x - x_old)
    x_old <- x

    # store momentum values
    f <- fn(par)
    fs[i] <- f
    all[i * 2] <- f
  }

  list(par = par, f = f, fs = fs, gd_fs = gd_fs, all = all)
}
```

In this routine, rather than store the velocity vector, we store the previous gradient descent result, `x_old`. In order to ensure the first iteration is gradient descent only, we also need to manually set the momentum coefficient `mu` to zero on the first iteration, which is what that `ifelse` expression does.

Also, there's some extra code to calculate and store the function values after the gradient descent stage. These aren't needed for the optimization to work, I just want to keep track of the values to demonstrate the different methods are in fact equivalent.

Sutskever Formulation

```
# Optimization by Sutskever Nesterov Momentum
#
# Return list also contains mu_fs: function evaluated after momentum
# stage of each iteration; all: function evaluated after gradient descent
# stage and momentum stage, in order.
snag <- function(par, fn, gr, lr, mu, max_iter = 10) {
  v <- rep(0, length(par))

  fs <- rep(0, max_iter)
  mu_fs <- rep(0, max_iter)
  all <- rep(0, max_iter * 2)

  for (i in 1:max_iter) {
    # momentum stage and update parameters
    mu_step <- mu * v
    par <- par + mu_step

    # store momentum results
    f <- fn(par)
    mu_fs[i] <- f
    all[i * 2 - 1] <- f

    # gradient descent stage
    g <- gr(par)
    gd_step <- -lr * g

    # update and store velocity for next step
    par <- par + gd_step
    v <- mu_step + gd_step

    # store gradient descent results
    f <- fn(par)
    fs[i] <- f
    all[i * 2] <- f
  }

  list(par = par, f = f, fs = fs, mu_fs = mu_fs, all = all)
}
```

This one is pretty straight-forward.

Bengio Formulation

```
# Optimization by Bengio Nesterov Momentum
bnag <- function(par, fn, gr, lr, mu, max_iter = 10) {
  fs <- rep(0, max_iter)

  v <- rep(0, length(par))
  for (i in 1:max_iter) {
    g <- gr(par)
    par <- par + mu * mu * v - (1 + mu) * lr * g
    v <- mu * v - lr * g

    # store results
    f <- fn(par)
    fs[i] <- f
  }

  list(par = par, f = f, fs = fs)
}
```

Because this implementation only uses a constant momentum coefficient, we replace the $\mu_{t+1}\mu_t$ term with μ_t^2. You would probably have to using a pretty wacky non-constant momentum schedule where this approximation had any major effect anyway... except when the momentum is changing from zero to non-zero, that is, in which case that's the difference between some momentum and no momentum. We shall return to this point later.

Dozat Nesterov Momentum

```
# Optimization by Dozat's expression for Nesterov momentum
dnag <- function(par, fn, gr, lr, mu, max_iter = 10) {
  fs <- rep(0, max_iter)

  # mprime, the recursive variable
  mpt <- 0
  mumu <- mu * mu

  v <- rep(0, length(par))
  for (i in 1:max_iter) {
    g <- gr(par)

    s <- lr * -g
    v <- (1 + mu) * s + mumu * mpt
    par <- par + v

    # update mprime for next step
    mpt <- mu * mpt + s
    # store results
    f <- fn(par)
    fs[i] <- f
  }

  list(par = par, f = f, fs = fs)
}
```

18 December 2021: Dozat's version of Nesterov momentum. We should expect this to give the same result as NAG and Bengio's Nesterov momentum.

Alternative momentum NAG Expression

```
# Optimization by Nesterov Accelerated Gradient, using a momentum-style
# update expression.
mnag <- function(par, fn, gr, lr, mu, max_iter = 10) {
  fs <- rep(0, max_iter)

  v <- rep(0, length(par))
  for (i in 1:max_iter) {
    g <- gr(par)
    v <- mu * (v - lr * g) - lr * g
    par <- par + v

    # setup v for the next iteration by removing the old gradient contribution
    v <- v + lr * g

    # store results
    f <- fn(par)
    fs[i] <- f
  }

  list(par = par, f = f, fs = fs)
}
```

Finally, here is your humble author's expression for NAG, written as a momentum-style update. To differentiate from the Sutskever and Bengio versions of NAG, I'll refer to it as momentum-NAG or mNAG.

Testing with Rosenbrock

Tradition dictates that I must demonstrate the use of these optimizers using the 2D Rosenbrock function, with a specific starting point:

```
par <- c(-1.2, 1)

fn <- function(x) {
  x1 <- x[1]
  x2 <- x[2]
  100 * (x2 - x1 * x1) ^ 2 + (1 - x1) ^ 2
}
gr <- function(x) {
  x1 <- x[1]
  x2 <- x[2]
  c(
    -400 * x1 * (x2 - x1 * x1) - 2 * (1 - x1),
    200 *      (x2 - x1 * x1))
}
```

Let's run the optimizers for 100 iterations. The point here is not whether we get really amazing optimization (we don't), but whether the outputs of the Sutskever and Bengio algorithms are equivalent. It would be a bonus if my mNAG result also worked. For comparison we'll throw in the classical momentum and as a sanity check, the vanilla NAG routine.

```
lr <- 0.001
mu <- 0.95
max_iter <- 100
snag_opt <- snag(par, fn, gr, lr, mu, max_iter)
bnag_opt <- bnag(par, fn, gr, lr, mu, max_iter)
nag_opt <- nag(par, fn, gr, lr, mu, max_iter)
mnag_opt <- mnag(par, fn, gr, lr, mu, max_iter)
dnag_opt <- dnag(par, fn, gr, lr, mu, max_iter)

cm_opt <- cm(par, fn, gr, lr, mu, max_iter)

sbnag_df <- data.frame(Bengio = bnag_opt$fs,
                 Sutskever = snag_opt$fs,
                 "Suts Mom" = snag_opt$mu_fs,
                 "NAG" = nag_opt$fs,
                 "mNAG" = mnag_opt$fs,
                 "Dozat" = dnag_opt$fs,
                 "CM" = cm_opt$fs)
```

Let's have a look at the first few iteration results:

```
knitr::kable(head(sbnag_df), caption = paste(
  "First few evaluations of NAG implementations, with lr = ",
  formatC(lr), "mu = ", formatC(mu),
  collapse = " "))
```

First few evaluations of NAG implementations, with lr = 0.001 mu = 0.95

Bengio	Sutskever	Suts.Mom	NAG	mNAG	Dozat	CM
34.960154	5.352912	24.200000	5.352912	34.960154	34.960154	5.352912
7.039334	6.144886	34.960154	5.257042	7.039334	7.039334	25.541437
5.020041	4.002116	7.039334	4.133512	5.020041	5.020041	22.488091
3.845406	3.895340	5.020041	4.096729	3.845406	3.845406	4.321887
3.746363	3.818670	3.845406	4.069433	3.746363	3.746363	18.826923
3.668701	3.741945	3.746363	4.050225	3.668701	3.668701	17.960659

The first two columns pit the Sutskever vs Bengio formulations directly. And, as we would expect, they're not the same: the Sutskever iteration result is from the gradient descent stage, and the Bengio result comes from the momentum stage. But if we put the momentum results from the Sutskever formulation up, we can see that they are the same as the Bengio result, but behind by one iteration, i.e. the Bengio result at iteration t matches the Sutskever momentum stage result at $t + 1$. The final column indicates the results using the alternative derivation I provided. It matches the Bengio results. Hurrah.

18 December 2021 The results for Dozat's version of Nesterov momentum have also been added. As expected, it matches the Bengio and mNAG results. Not much also to say about it so it won't appear in any further discussion.

However, none of the results match the vanilla NAG implementation. A clue to what's going on is from looking at the classical momentum result in the "CM" column. The first iteration of classical momentum is always plain gradient descent, and that's also how NAG works by construction. We can see that both NAG and CM have the same result on the first iteration, which suggests they're working correctly. The Sutskever version is also doing gradient descent on its first step. But the other implementations clearly aren't doing gradient descent on their first iteration, and on subsequent iterations even the Sutskever version diverges from the NAG result.

We'll get back to this, but let's just make sure these observations holds up at the end of the table too.

```
knitr::kable(tail(sbnag_df), caption = paste(
  "Last few evaluations of NAG implementations, with lr = ",
  formatC(lr), "mu = ", formatC(mu),
  collapse = " "))
```

Last few evaluations of NAG implementations, with lr = 0.001 mu = 0.95

	Bengio	Sutskever	Suts.Mom	NAG	mNAG	Dozat	CM
95	0.0142850	0.0151685	0.0151835	0.0590959	0.0142850	0.0142850	0.4448498
96	0.0134463	0.0142710	0.0142850	0.0552234	0.0134463	0.0134463	0.5019262
97	0.0126630	0.0134332	0.0134463	0.0516306	0.0126630	0.0126630	0.4306990
98	0.0119314	0.0126508	0.0126630	0.0482955	0.0119314	0.0119314	0.3112414
99	0.0112476	0.0119199	0.0119314	0.0451983	0.0112476	0.0112476	0.2869773
100	0.0106082	0.0112368	0.0112476	0.0423208	0.0106082	0.0106082	0.3721692

The Sutskever, Bengio and momentum NAG implementations all still match up in the same way they did. CM and NAG are off doing their own thing. I suppose it's at least reassuring that for this arbitrarily chosen set of learning rate and momentum coefficient values, the NAG results all do better than classical momentum. And in fact it seems like not doing plain gradient descent on the first iteration gives better results, at least for this set of parameters.

Behavior on first iteration

The reason for the differing behaviors of the NAG implementations is down to two things:

- For some implementations (the Bengio and momentum NAG formulations), simply setting the initial velocity vector, v_0, to zero isn't enough to get gradient descent on the first iteration.
- The Sutskever version of NAG does a different pattern of gradient descent and momentum stages in its iterations, compared to vanilla NAG.

For the first point, the implementation of Bengio Nesterov momentum and momentum NAG are both giving the "long" version of the steepest descent on the first iteration. In the discussion on unrolling the updates where we first came across this point of difference with CM, I decided that the "long" version of the first iteration was the right decision, but you can see that it causes some extra issues in our comparisons.

What about the Sutskever formulation? Let's think about the chain of parameter updates that actually take place over the first few iterations:

For standard NAG, the chain is: gradient descent stage, momentum stage, gradient descent stage, momentum stage. Except the momentum stage results in zero change on iteration one because of the zero velocity vector, so what actually happens is: gradient descent stage, gradient descent stage, momentum stage. Let's call that `g|gm` for short with the bar indicating where the end of an iteration occurs. To make the pattern more obvious, the first three iterations look like `g|gm|gm`. This is the same pattern as classical momentum, although the `m` stages are obviously different in content.

What about the Sutskever formulation? Its pattern for the first two iterations is: momentum stage, gradient descent stage, momentum stage, gradient descent stage. Except, once again, the momentum stage doesn't happen on the first iteration, so you actually get: gradient descent stage, momentum stage, gradient descent stage, and so on. So for Sutskever momentum, the first three iterations look like `g|mg|mg`.

Therefore, as you can see, the initial set of gradient and momentum stages is different for the Sutskever formulation, which is able to strictly interleave gradient descent and momentum stages, whereas NAG begins, like classical momentum, with two gradient descent stages. This difference shouldn't be too surprising, because as we saw when deriving the Sutskever formulation, we conveniently "forget" about an initial gradient descent stage compared to NAG. Anyway, all this means that the parameters end up in different locations after the first iteration.

Consistent NAG momentum algorithms

It's not clear to me that there is any particular theoretical or practical benefit to trying to bend these different formulations to give either the same gradient descent on the first iteration as CM, or to having them give the same pattern as vanilla NAG. But given the effort that's been made to show that NAG is like classical momentum in some respects, we may as well see what it would take.

Sutskever Formulation consistent with NAG

```
# Sutskever Nesterov Momentum consistent with NAG
#
# Extra parameter wait: wait this number of extra iterations before applying
# momentum. Needed only to sync up with other implementations of nesterov
# momentum: set wait to 1 to make mu_f at iter i match the output of the
# other implementations at iter i-1.
snagc <- function(par, fn, gr, lr, mu, max_iter = 10, wait = 0) {
  v <- rep(0, length(par))

  fs <- rep(0, max_iter)
  mu_fs <- rep(0, max_iter)
  all <- rep(0, max_iter * 2)

  for (i in 1:max_iter) {
    # momentum stage and update parameters
    mu_step <- ifelse(i > wait + 1, mu, 0) * v
    par <- par + mu_step

    # store momentum results
    f <- fn(par)
    mu_fs[i] <- f
    all[i * 2 - 1] <- f

    # gradient descent stage
    g <- gr(par)
    gd_step <- -lr * g

    # update and store velocity for next step
    par <- par + gd_step
    v <- mu_step + gd_step

    # store gradient descent results
    f <- fn(par)
    fs[i] <- f
    all[i * 2] <- f
  }

  list(par = par, f = f, fs = fs, mu_fs = mu_fs, all = all)
}
```

The Sutskever formulation already does gradient descent on its first iteration, so there's not a huge change required for the algorithm. Instead, I've introduced a new parameter `wait`, that indicates the number of iterations to wait before applying momentum. If this is set to `0`, then you get the current behavior. Set it to `1` and the momentum step will be zero on the second iteration, too. That gives us the two gradient descent stages in a row that should sync us up with NAG.

Bengio Formulation consistent with NAG

```
# Bengio Nesterov Momentum consistent with NAG
bnagc <- function(par, fn, gr, lr, mu, max_iter = 10) {
  fs <- rep(0, max_iter)

  v <- rep(0, length(par))
  for (i in 1:max_iter) {
    g <- gr(par)
    if (i == 1) {
      par <- par - lr * g
    }
    else {
      par <- par + (mu * mu * v) - ((1 + mu) * lr * g)
      v <- (mu * v) - (lr * g)
    }

    # store gradient descent results
    f <- fn(par)
    fs[i] <- f
  }

  list(par = par, f = f, fs = fs)
}
```

"Fixing" the Bengio formulation turns out to be a bit more work than you might think. As we noticed earlier, to get gradient decent on the first iteration, μ_1, needs to be set to zero. However, as a result this means the assumption we made that $\mu_t \mu_{t-1} = \mu_{t-1}^2$ isn't correct for iteration 2: $\mu_2 \mu_1 = 0 \neq \mu_2^2$. However, we can't just set `mu` to 0 for the second iteration, because the non-zero μ_2 is used on its own as part of the gradient descent expression.

There are two ways to deal with this: start storing the value of `mu` from the previous iteration, even though I purposely chose a constant value of `mu` to keep things simple, or just force v_1 to be zero by not updating it on the first iteration. I chose the latter.

Momentum NAG consistent with NAG

```
# momentum NAG consistent with NAG
mnagc <- function(par, fn, gr, lr, mu, max_iter = 10) {
  fs <- rep(0, max_iter)

  v <- rep(0, length(par))
  for (i in 1:max_iter) {
    g <- gr(par)
    v <- (ifelse(i == 1, 0, mu) * (v - lr * g)) - lr * g
    par <- par + v

    # setup v for the next iteration by removing the old gradient contribution
    v <- v + lr * g

    # store results
    f <- fn(par)
    fs[i] <- f
  }

  list(par = par, f = f, fs = fs)
}
```

And last, the rewritten momentum NAG update. This required the least modification from the original routine: simply set the momentum coefficient to zero on the first iteration.

Results for the “NAG-consistent” routines

Time to look at some numbers again. We’ll compare the new versions of the Sutskever, Bengio and Momentum formulations of NAG with vanilla NAG. We’ll also once again pull out the momentum stage results for Sutskever so we can compare directly to the Bengio result.

```
mnagc_opt <- mnagc(par, fn, gr, lr, mu, max_iter)
snagc_opt <- snagc(par, fn, gr, lr, mu, max_iter, wait = 1)
bnagc_opt <- bnagc(par, fn, gr, lr, mu, max_iter)

ncdf <- data.frame(cBengio = bnagc_opt$fs,
                   cSutskever = snagc_opt$fs,
                   "Suts Mom" = snagc_opt$mu_fs,
                   NAG = nag_opt$fs,
                   cmNAG = mnagc_opt$fs,
                   "NAG gd" = nag_opt$gd_fs)
```

Let’s have a look at the first few iteration results:

```
knitr::kable(head(ncdf), caption = paste(
  "First few evaluations of NAG consistent implementations, with lr = ",
  formatC(lr), "mu = ", formatC(mu),
  collapse = " "))
```

First few evaluations of NAG consistent implementations, with lr = 0.001 mu = 0.95

cBengio	cSutskever	Suts.Mom	NAG	cmNAG	NAG.gd
5.352912	5.352912	24.200000	5.352912	5.352912	5.352912
5.257042	4.117790	5.352912	5.257042	5.257042	4.117790
4.133512	4.118466	5.257042	4.133512	4.133512	4.118466
4.096729	4.097143	4.133512	4.096729	4.096729	4.097143
4.069433	4.082870	4.096729	4.069433	4.069433	4.082870
4.050225	4.066122	4.069433	4.050225	4.050225	4.066122

These are in the same order as the previous table. The Bengio and Sutskever formulations don't match, as expected. But the Sutskever momentum result in the third column *does* match the Bengio results from the previous iteration. So relative performance between Sutskever and Bengio has been maintained. But in addition, the NAG result in the fourth column matches the Bengio and Sutskever momentum results. The momentum NAG result has also been correctly updated. Finally, as an extra check, the NAG gradient desent result is shown in the final column, which also matches up with the Sutskever results.

And let's take a look at the final few iterations, just to make sure everything still holds up:

```
knitr::kable(tail(ncdf), caption = paste(
  "Last few evaluations of NAG consistent implementations, with lr = ",
  formatC(lr), "mu = ", formatC(mu),
  collapse = " "))
```

Last few evaluations of NAG consistent implementations, with lr = 0.001 mu = 0.95

	cBengio	cSutskever	Suts.Mom	NAG	cmNAG	NAG.gd
95	0.0590959	0.0631888	0.0632715	0.0590959	0.0590959	0.0631888
96	0.0552234	0.0590202	0.0590959	0.0552234	0.0552234	0.0590202
97	0.0516306	0.0551542	0.0552234	0.0516306	0.0516306	0.0551542
98	0.0482955	0.0515670	0.0516306	0.0482955	0.0482955	0.0515670
99	0.0451983	0.0482371	0.0482955	0.0451983	0.0451983	0.0482371
100	0.0423208	0.0451446	0.0451983	0.0423208	0.0423208	0.0451446

That's a relief.

Conclusions

Yes, NAG can be thought of as being like classical momentum but where you do the momentum step first and *then* the gradient step. But there's more to it than that.

Both classical momentum and NAG use a weighted combination of the current and past gradients to determine the next direction of descent. The difference is in the distribution of weights: NAG places more emphasis on recent gradients. So it might succeed better than classical momentum under conditions where the old gradients quickly become uniformative (regions of high curvature?). Personally, I find this view of NAG through the lens of a weighted history of gradients more appealing than the Sutskever formulation because it leads to interesting ideas like Quasi-Hyperbolic Momentum.

In terms of implementation, I pity anyone tasked with implementing Nesterov momentum and demonstrating that they actually got it right. Imagine naively implementing vanilla NAG, the Sutskever formulation and the Bengio formulation. Using identical test functions, starting points and parameters, their output are all different from each other *but they're all correct*!