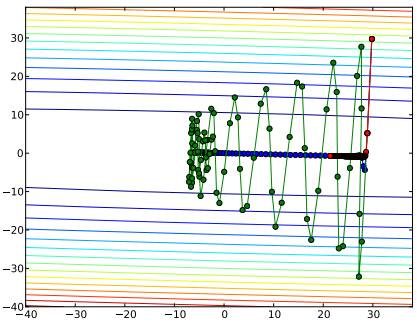

Figure 2. The trajectories of CM, NAG, and SGD are shown. Although the value of the momentum is identical for both experiments, CM exhibits oscillations along the high-curvature directions, while NAG exhibits no such oscillations. The global minimizer of the objective is at (0,0). The red curve shows gradient descent with the same learning rate as NAG and CM, the blue curve shows NAG, and the green curve shows CM. See section 2 of the paper.

Appendix

A.1 Derivation of Nesterov's Accelerated Gradient as a Momentum Method

Nesterov's accelerated gradient is an iterative algorithm that was originally derived for non-stochastic gradients. It is initialized by setting $k = 0$, $a_0 = 1$, $\theta_{-1} = y_0$, y_0 to an arbitrary parameter setting, z to an arbitrary parameter setting, and $\varepsilon_{-1} = \|y_0 - z\|/\|\nabla f(y_0) - \nabla f(z)\|$. It then repeatedly updates its parameters with the following equations:

$$\varepsilon_t = 2^{-i}\varepsilon_{t-1} \tag{6}$$

(here i is the smallest positive integer for which

$$f(y_t) - f(y_t - 2^{-i}\varepsilon_{t-1}\nabla f(y_t)) \geq 2^{-i}\varepsilon_{t-1}\frac{\|\nabla f(y_t)\|^2}{2})$$

$$\theta_t = y_t - \varepsilon_t \nabla f(y_t) \tag{7}$$

$$a_{t+1} = \left(1 + \sqrt{4a_t^2 + 1}\right)/2 \tag{8}$$

$$y_{t+1} = \theta_t + (a_t - 1)(\theta_t - \theta_{t-1})/a_{t+1} \tag{9}$$

The above presentation is relatively opaque and could be difficult to understand, so we will rewrite these equations in a more intuitive manner.

The learning rate ε_t is adapted to always be smaller than the reciprocal of the "observed" Lipshitz coefficient of ∇f around the trajectory of the optimization. Alternatively, if the Lipshitz coefficient of the derivative is known to be equal to L, then setting $\varepsilon_t = 1/L$ for all t is sufficient to obtain the same theoretical guarantees. This method for choosing the learning rate assumes that f is not noisy, and will result in too-large learning rates if the objective is stochastic.

To understand the sequence a_t, we note that the function $x \to \left(1 + \sqrt{4x^2 + 1}\right)/2$ quickly approaches $x \to x + 0.5$ from below as $x \to \infty$, so $a_t \approx (t + 4)/2$ for large t, and thus $(a_t - 1)/a_{t+1}$ (from eq. 9) behaves like $1 - 3/(t + 5)$.

Finally, if we define

$$v_t \equiv \theta_t - \theta_{t-1} \tag{10}$$

$$\mu_t \equiv (a_t - 1)/a_{t+1} \tag{11}$$

then the combination of eqs. 9 and 11 implies:

$$y_t = \theta_{t-1} + \mu_{t-1}v_{t-1}$$

which can be used to rewrite eq. 7 as follows:

$$\theta_t = \theta_{t-1} + \mu_{t-1}v_{t-1} - \varepsilon_{t-1}\nabla f(\theta_{t-1} + \mu_{t-1}v_{t-1}) \tag{12}$$

$$v_t = \mu_{t-1}v_{t-1} - \varepsilon_{t-1}\nabla f(\theta_{t-1} + \mu_{t-1}v_{t-1}) \tag{13}$$

where eq. 13 is a consequence of eq. 10. Alternatively:

$$v_t = \mu_{t-1}v_{t-1} - \varepsilon_{t-1}\nabla f(\theta_{t-1} + \mu_{t-1}v_{t-1}) \tag{14}$$

$$\theta_t = \theta_{t-1} + v_t \tag{15}$$

where $\mu_t \approx 1 - 3/(t + 5)$. (Nesterov, 1983) shows that if f is a convex function with an L-Lipshitz continuous derivative, then the above method satisfies the following:

$$f(\theta_t) - f(\theta^*) \leq \frac{4L\|\theta_{-1} - \theta^*\|^2}{(t + 2)^2} \tag{16}$$

To understand the quadratic speedup obtained by the momentum, consider applying momentum to a linear function. In this case, the i-th step of the momentum method will be of distance proportional to i; therefore N steps could traverse a quadratically longer distance: $1 + 2 + \cdots + N = O(N^2)$.

A.2 Details for Theorem 2.1

We will first formulate and prove a result which establishes the well known fact that first-order methods such CM and NAG are invariant to orthonormal transformations (i.e. rotations) such as U. In particular, we will will show that the sequence of iterates obtained by applying NAG and CM to the reparameterized quadratic p, is given by U times sequence of iterates obtained by applying NAG and CM to the original quadratic q. Note that the only fact we use about U at this stage is that it is orthonormal/unitary, not that it diagonalizes q.

Proposition 6.1. *Let $\{(\mu_i, \varepsilon_i)\}_{i=1}^{\infty}$ be an arbitrary sequence of learning rates, let x_0, v_0 be an arbitrary initial position and velocity, and let $\{(x_i, v_i)\}_{i=0}^{\infty}$ be the sequence of iterates obtained by CM by optimizing q starting from x_0, v_0, with the learning parameters μ_i, ε_i at iteration i.*

Next, let y_0, w_0 be given by Ux_0, Uv_0, and let $\{(y_i, w_i)\}_{i=0}^{\infty}$ be the sequence of iterates obtained by CM by optimizing p starting from y_0, w_0, with the learning parameters μ_i, ε_i at iteration i.

Then the following holds for all i:

$$y_i = Ux_i$$
$$w_i = Uv_i$$

The above also applies when CM is replaced with NAG.

Proof. First, notice that

$$\begin{aligned} x_{i+1} &= CM_{\mathrm{x}}(\mu_i, q, x_i, v_i) \\ v_{i+1} &= CM_{\mathrm{v}}(\mu_i, q, x_i, v_i) \end{aligned}$$

and that

$$\begin{aligned} y_{i+1} &= CM_{\mathrm{x}}(\mu_i, p, y_i, w_i) \\ w_{i+1} &= CM_{\mathrm{v}}(\mu_i, p, y_i, w_i) \end{aligned}$$

The proof is by induction. The claim is immediate for $i = 0$. To see that it holds for $i + 1$ assuming i, consider:

$$\begin{aligned} w_{i+1} &= w_i + \varepsilon \nabla_{y_i} p(y_i) \\ &= w_i + \varepsilon U \nabla_{x_i} p(y_i) && \text{(chain rule using } y_i = Ux_i) \\ &= w_i + \varepsilon U \nabla_{x_i} q(U^\top y_i) && \text{(definition of } p) \\ &= w_i + \varepsilon U \nabla_{x_i} q(x_i) && (x_i = U^\top y_i) \\ &= Uv_i + \varepsilon U \nabla_{x_i} q(x_i) && \text{(by induction)} \\ &= U(v_i + \varepsilon \nabla_{x_i} q(x_i)) \\ &= Uv_{i+1} \end{aligned}$$

Using the above, we get

$$\begin{aligned} y_{i+1} &= y_i + w_{i+1} \\ &= Ux_i + Uv_{i+1} \\ &= Ux_{i+1} \end{aligned}$$

$$y_{i+1} = y_i + w_{i+1} = Ux_i + Uv_{i+1} = Ux_{i+1}$$

This completes the proof for CM; the proof for NAG is nearly identical.

□

Given this result, and reparameterization p of q according to its eigencomponents, the results of Theorem 2.1 can thus be

Proof of Theorem 2.1. We first show that for separable problems, CM (or NAG) is precisely equivalent to many simultaneous applications of CM (or NAG) to the one-dimensional problems that correspond to each problem dimension. We then show that for NAG, the effective momentum for a one-dimensional problem depends on its curvature. We prove these results for one step of CM or NAG, although these results generalize to larger n.

Consider one step of CM_{v}:

$$\begin{aligned} &CM_{\mathrm{v}}(\mu, p, y, v) = \mu v - \varepsilon \nabla p(y) \\ &= (\mu[v]_1 - \varepsilon \nabla_{[y]_1} p(y), \ldots, \mu[v]_n - \varepsilon \nabla_{[y]_n} p(y)) \\ &= (\mu[v]_1 - \varepsilon \nabla [p]_1([y]_1), \ldots, \mu[v]_n - \varepsilon \nabla [p]_n([y]_n)) \\ &= (CM_{\mathrm{v}}(\mu, [p]_1, [y]_1, [v]_1), \ldots, CM_{\mathrm{v}}(\mu, [p]_n, [y]_n, [v]_n)) \end{aligned}$$

This shows that one step of CM_{v} on q is precisely equivalent to n simultaneous applications of CM_{v} to the one-dimensional quadratics $[q]_i$, all with the same μ and ε. A similar argument shows a single step of each of CM_{x}, on NAG_{x}, and NAG_{v} can be obtained by applying each of them to the n one-dimensional quadratics $[q]_i$.

Next we show that NAG, applied to a one-dimensional quadratic with a momentum coefficient μ, is equivalent to CM applied to the same quadratic and with the same learning rate, but with a momentum coefficient $\mu(1 - \varepsilon\lambda)$. We show this by expanding $NAG_{\mathrm{v}}(\mu, [p]_i, y, v)$ (where y and v are scalars):

$$\begin{aligned} NAG_{\mathrm{v}}(\mu, [q]_i, y, v) &= \mu v - \varepsilon \nabla [p]_i(y + \mu v) \\ &= \mu v - \varepsilon(\lambda_i(y + \mu v) + c_i) \\ &= \mu v - \varepsilon \lambda_i \mu v - \varepsilon(\lambda_i y + c_i) \\ &= \mu(1 - \varepsilon\lambda_i)v - \varepsilon \nabla [p]_i(y) \\ &= CM_{\mathrm{v}}(\mu(1 - \varepsilon\lambda_i), [p]_i, v, y) \end{aligned}$$

Since Eq. 2 is identical to 4, it follows that

$$NAG_{\mathrm{x}}(\mu, [p]_i, y, v) = CM_{\mathrm{x}}(\mu(1 - \varepsilon\lambda_i), [p]_i, y, v)$$

for all i.

□

A.3. Autoencoder Problem Details

We experiment with the three autoencoder problems from Hinton & Salakhutdinov (2006), which are described in Table 6.

A.4. RNN Problem Details

We considered 4 of the pathological long term dependency "problems" from Hochreiter & Schmidhuber (1997), which consist of artificial datasets designed to have various non-trivial long-range dependencies.

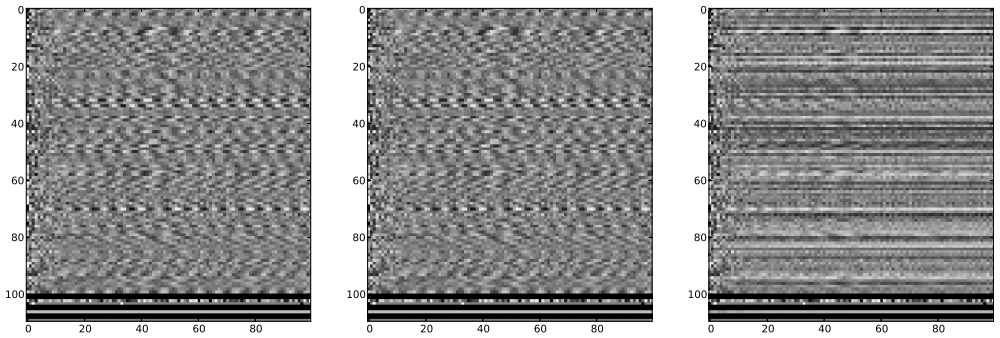

Figure 3. In each figure, the vertical axis is time, and the horizontal axis are units. The leftmost figure shows the hidden state sequences of an RNN on the addition problem with the aforementioned parameter initialization. Note the "gentle" oscillations of the hidden states. The middle figure shows the hidden state sequence of the same RNN after 1000 parameter updates. The hidden state sequence is almost indistinguishable, differing by approximately 10^{-4} for each pixel, so despite the little progress that has been made, the oscillations are preserved, and thus the parameter setting still have a chance of solving the problem. The rightmost figure shows an RNN with the same initialization after 1000 parameter updates, where the output biases were initialized to zero. Notice how the hidden state sequence has many fewer oscillations, although both parameter settings fail to establish communication between the inputs and the target.

name	dim	size	architecture
Curves	784	20,000	784-400-200-100-50-25-6
Mnist	784	60,000	784-1000-500-250-30
Faces	625	103,500	625-2000-1000-500-30

Table 6. The networks' architectures and the sizes of the datasets.

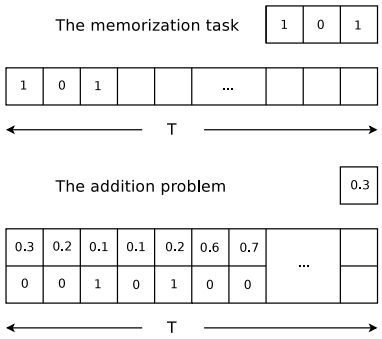

Figure 4. The memorization and the addition problems. The goal of the memorization problem is to output the input bits in the correct order. The goal of the addition and the multiplication problem is to output the sum (or the product) of the two marked inputs.

These were the 5-bit memorization task, the 20-bit memorization task, the addition problem, and the multiplication problem. These artificial problems were each designed to be impossible to learn with regular RNNs using standard optimization methods, owing to the presence of long-range temporal dependencies of the target outputs on the early inputs. And in particular, despite the results of **?**, they cannot be learned with ESNs that have 100 conventional hidden units (Jaeger, 2012)[1].

In the 5-bit memorization problem, the input sequence consists of 5 bits that are followed by a large number of blank symbols. The target sequence consists of the same 5 bits occurring at the end of the sequence, preceded by blanks. The 20-bit memorization problem is similar to the 5-bit memorization problem, but the 5-bits are replaced with ten 5-ary symbols, so that each sequence contains slightly more than 20 bits of information. In the addition problem, the input sequence consists of pairs (x, y) of real numbers presented in sequence, where each x is a drawn from $U[0, 1]$ and each y is a binary "marker" in $\{0, 1\}$. The final entry of the target output sequence is determined as the sum of the x component for the two pairs whose y component is 1. The multiplication problem is analogous, and we follow the precise format used by Martens & Sutskever (2011). To achieve low error on each of these tasks the RNN must learn to memorize and transform some information contained at the beginning of the sequence

[1]Small ESNs that use leaky integration can achieve very low training errors on these problems, but the leaky integration is well-suited for the addition and the multiplication but not the memorization problems.

within its hidden state, retaining it all of the way to the end. This is made especially difficult because there are no easier-to-learn short or medium-term dependencies that can act as hints to help the learning see the long-term ones, and because of the noise in the addition and the multiplication problems.

A.5 The effect of bias centering

In our experiments, we observed that the hidden state oscillations, which are likely important for relaying information across long distances, had a tendency to disappear after the early stages of learning (see fig. 3), causing the optimization to converge to a poor local optimum. We surmised that this tendency was due to fact that the smallest modification to the parameters (as measured in the standard norm - the quantity which steepest descent can be thought of as minimizing) that allowed the output units to predict the average target output, involved making the outputs constant by way of making the hidden state constant. By centering the target outputs, the initial bias is correct by default, and thus the optimizer is not forced into making this "poor early choice" that dooms it later on.

Similarly, we found it necessary to center the *inputs* to reliably solve the multiplication problem; the momentum methods were unable to solve this problem without input centering. We speculate that the reason for this is that the inputs associated with the multiplication problem are positive, causing the hidden state to drift in the direction of the input-to-hidden vector, which in turn leads to saturation of the hidden state. When the inputs are centered, the net input to the hidden units will initial be closer to zero when averaged over a reasonable window of time, thus preventing this issue.

Figure 3 shows how the hidden state sequence evolves when the output units are not centered. While the oscillations of the RNN's hidden state are preserved if the output bias is initialized to be the mean of the targets, they disappear if the output bias is set to zero (in this example, the mean of the targets is 0.5). This is a consequence of the isotropic nature of stochastic gradient descent which causes it to minimize the L_2 distance of its optimization paths, and of the fact that the L_2-closest parameter setting that outputs the average of the targets is obtained by slightly adjusting both the biases and making the hidden states constant, to utilize the hidden-to-output weights.

A.6 Hybrid HF-Momentum approach

Our hybrid HF-momentum algorithm, is characterized by the following changes to the original approach outlined by Martens (2010): we disposed of the line search and used a fixed learning rate of 1 which, after some point chosen by hand, we gradually decayed to zero using a simple schedule; we gradually increased the value of the decay constant (similarly to how we increased μ); and we used smaller minibatches, while computing the gradient only on the current minibatch (as opposed to the full dataset - as is commonly done with HF). Analogously to our experiments with momentum, we adjusted the settings near the end of optimization (after the transient phase) to improve fine-grained local convergence/stochastic estimation. In particular, we switched back to using a more traditional version of HF along the lines of the one described by Martens (2010), which involved using much larger minibatches, or just computing the gradient on more data/the entire training set, raising the learning rate back to 1 (or as high as the larger minibatches permit), etc., and also decreasing the L2 weight decay to squeeze out some extra performance.

The results of these experiments were encouraging, although preliminary, and we report them in Table 1, noting that they are not directly comparable to the experiments performed with CM and NAG due to the of use incomparable amounts of computation (fewer iterations, but with large minibatches).

For each autoencoder training task we ran we hand-designed a schedule for the learning rate (ε), decay constant (μ), and minibatch size (s), after a small amount of trial and error. The details for CURVES and MNIST are given below.

For CURVES, we first ran HF iterations with 50 CG steps each, for a total of 250k CG steps, using $\varepsilon = 1.0$, $\mu = 0.95$ and with gradient and curvature products computed on minibatches consisting of 1/16th of the training set. Next, we increased μ to 0.999 and annealed ε according to the schedule $500/(t - 4000)$. The error reach 0.0094 by this point, and from here we tuned the method to achieve fine local convergence, which we achieved primarily through running the method in full batch mode. We first ran HF for 100k total CG steps with the number of CG steps per update increased to 200, μ lowered to 0.995, and ε raised back to 1 (which was stable because we were running in batch mode) . The error reach 0.087 by this point. Finally, we lowered the L2 weight decay from 2e-5 (which was used in (Martens, 2010)) to near zero and ran 500k total steps more, to arrive at an error of 0.058.

For MNIST, we first ran HF iterations with 50 CG steps each, for a total of 25k CG steps, using $\varepsilon = 1.0$, $\mu = 0.95$ and with gradient and curvature products computed on minibatches consisting of 1/20th of the training set. Next, we increased μ to 0.995, ε was annealed according to $250/t$, and the method was run for another 225k CG steps. The error reach 0.81 by this

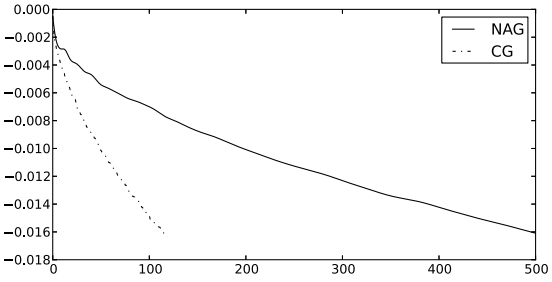

Figure 5. Comparison of NAG and CG on a damped quadratic objective from the middle of HF. The plot shows that while CG is considerably more effective than NAG at opitimizing the kinds of quadratic approximation that occur during neural network learning, although the difference is not dramatic. Note that we truncated CG after 120 steps because this is the number of steps that would be typically used by HF at this stage of learning.

point. Finally, we focused on achieving fine convergence and started computing the gradient on the full training set (but still computing the curvature products on minibatches of size 1/20th). With ε set back to 1, we ran another 125k total CG steps to achieve a final error of 0.69. The L2 weight decay was set to 1e-5 for the entirety of training.