On the importance of initialization and momentum in deep learning

Ilya Sutskever[1] ILYASU@GOOGLE.COM
James Martens JMARTENS@CS.TORONTO.EDU
George Dahl GDAHL@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU

Abstract

Deep and recurrent neural networks (DNNs and RNNs respectively) are powerful models that were considered to be almost impossible to train using stochastic gradient descent with momentum. In this paper, we show that when stochastic gradient descent with momentum uses a well-designed random initialization and a particular type of slowly increasing schedule for the momentum parameter, it can train both DNNs and RNNs (on datasets with long-term dependencies) to levels of performance that were previously achievable only with Hessian-Free optimization. We find that both the initialization and the momentum are crucial since poorly initialized networks cannot be trained with momentum and well-initialized networks perform markedly worse when the momentum is absent or poorly tuned.

Our success training these models suggests that previous attempts to train deep and recurrent neural networks from random initializations have likely failed due to poor initialization schemes. Furthermore, carefully tuned momentum methods suffice for dealing with the curvature issues in deep and recurrent network training objectives without the need for sophisticated second-order methods.

我们展示SGD带有动量的+随机初始化+特别类型的小的增加动量参数=可以良好训练DNN和RNN

1. Introduction

Deep and recurrent neural networks (DNNs and RNNs, respectively) are powerful models that achieve high performance on difficult pattern recognition problems in vision, and speech (Krizhevsky et al., 2012; Hinton et al., 2012; Dahl et al., 2012; Graves, 2012).

Although their representational power is appealing, the difficulty of training DNNs has prevented their widepread use until fairly recently. DNNs became the subject of renewed attention following the work of Hinton et al. (2006) who introduced the idea of greedy layerwise pre-training. This approach has since branched into a family of methods (Bengio et al., 2007), all of which train the layers of the DNN in a sequence using an auxiliary objective and then "fine-tune" the entire network with standard optimization methods such as stochastic gradient descent (SGD). More recently, Martens (2010) attracted considerable attention by showing that a type of truncated-Newton method called Hessian-free Optimization (HF) is capable of training DNNs from certain random initializations without the use of pre-training, and can achieve lower errors for the various auto-encoding tasks considered by Hinton & Salakhutdinov (2006).

Recurrent neural networks (RNNs), the temporal analogue of DNNs, are highly expressive sequence models that can model complex sequence relationships. They can be viewed as very deep neural networks that have a "layer" for each time-step with parameter sharing across the layers and, for this reason, they are considered to be even harder to train than DNNs. Recently, Martens & Sutskever (2011) showed that the HF method of Martens (2010) could effectively train RNNs on artificial problems that exhibit very long-range dependencies (Hochreiter & Schmidhuber, 1997). Without resorting to special types of memory units, these problems were considered to be impossibly difficult for first-order optimization methods due to the well known vanishing gradient problem (Bengio et al., 1994). Sutskever et al. (2011) and later Mikolov et al. (2012) then applied HF to train RNNs to perform character-level language modeling and achieved excellent results.

Recently, several results have appeared to challenge the commonly held belief that simpler first-order methods are incapable of learning deep models from random initializations. The work of Glorot & Bengio (2010), Mohamed et al. (2012), and Krizhevsky et al. (2012) reported little difficulty training neural networks with depths up to 8 from certain well-chosen

Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28.

[1]Work was done while the author was at the University of Toronto.

random initializations. Notably, Chapelle & Erhan (2011) used the random initialization of Glorot & Bengio (2010) and SGD to train the 11-layer autoencoder of Hinton & Salakhutdinov (2006), and were able to surpass the results reported by Hinton & Salakhutdinov (2006). While these results still fall short of those reported in Martens (2010) for the same tasks, they indicate that learning deep networks is not nearly as hard as was previously believed.

The first contribution of this paper is a much more thorough investigation of the difficulty of training deep and temporal networks than has been previously done. In particular, we study the effectiveness of SGD when combined with well-chosen initialization schemes and various forms of momentum-based acceleration. We show that while a definite performance gap seems to exist between plain SGD and HF on certain deep and temporal learning problems, this gap can be eliminated or nearly eliminated (depending on the problem) by careful use of classical momentum methods or Nesterov's accelerated gradient. In particular, we show how certain carefully designed schedules for the constant of momentum μ, which are inspired by various theoretical convergence-rate theorems (Nesterov, 1983; 2003), produce results that even surpass those reported by Martens (2010) on certain deep-autencoder training tasks. For the long-term dependency RNN tasks examined in Martens & Sutskever (2011), which first appeared in Hochreiter & Schmidhuber (1997), we obtain results that fall just short of those reported in that work, where a considerably more complex approach was used.

Our results are particularly surprising given that momentum and its use within neural network optimization has been studied extensively before, such as in the work of Orr (1996), and it was never found to have such an important role in deep learning. One explanation is that previous theoretical analyses and practical benchmarking focused on local convergence in the stochastic setting, which is more of an estimation problem than an optimization one (Bottou & LeCun, 2004). In deep learning problems this final phase of learning is not nearly as long or important as the initial "transient phase" (Darken & Moody, 1993), where a better argument can be made for the beneficial effects of momentum.

In addition to the inappropriate focus on purely local convergence rates, we believe that the use of poorly designed standard random initializations, such as those in Hinton & Salakhutdinov (2006), and suboptimal meta-parameter schedules (for the momentum constant in particular) has hampered the discovery of the true effectiveness of first-order momentum methods in deep learning. We carefully avoid both of these pitfalls in our experiments and provide a simple to understand and easy to use framework for deep learning that is surprisingly effective and can be naturally combined with techniques such as those in Raiko et al. (2011).

We will also discuss the links between classical momentum and Nesterov's accelerated gradient method (which has been the subject of much recent study in convex optimization theory), arguing that the latter can be viewed as a simple modification of the former which increases stability, and can sometimes provide a distinct improvement in performance we demonstrated in our experiments. We perform a theoretical analysis which makes clear the precise difference in local behavior of these two algorithms. Additionally, we show how HF employs what can be viewed as a type of "momentum" through its use of special initializations to conjugate gradient that are computed from the update at the previous time-step. We use this property to develop a more momentum-like version of HF which combines some of the advantages of both methods to further improve on the results of Martens (2010).

2. Momentum and Nesterov's Accelerated Gradient

The momentum method (Polyak, 1964), which we refer to as classical momentum (CM), is a technique for accelerating gradient descent that accumulates a velocity vector in directions of persistent reduction in the objective across iterations. Given an objective function $f(\theta)$ to be minimized, classical momentum is given by:

$$v_{t+1} = \mu v_t - \varepsilon \nabla f(\theta_t) \quad (1)$$

$$\theta_{t+1} = \theta_t + v_{t+1} \quad (2)$$

where $\varepsilon > 0$ is the learning rate, $\mu \in [0, 1]$ is the momentum coefficient, and $\nabla f(\theta_t)$ is the gradient at θ_t.

Since directions d of low-curvature have, by definition, slower local change in their rate of reduction (i.e., $d^\top \nabla f$), they will tend to persist across iterations and be amplified by CM. Second-order methods also amplify steps in low-curvature directions, but instead of accumulating changes they reweight the update along each eigen-direction of the curvature matrix by the inverse of the associated curvature. And just as second-order methods enjoy improved local convergence rates, Polyak (1964) showed that CM can considerably accelerate convergence to a local minimum, requiring $\sqrt{R}$-times fewer iterations than steepest descent to reach the same level of accuracy, where R is the condition number of the curvature at the minimum and μ is set to $(\sqrt{R}-1)/(\sqrt{R}+1)$.

Nesterov's Accelerated Gradient (abbrv. NAG; Nesterov, 1983) has been the subject of much recent attention by the convex optimization community (e.g., Cotter et al., 2011; Lan, 2010). Like momentum, NAG is a first-order optimization method with better convergence rate guarantee than gradient descent in

certain situations. In particular, for general smooth (non-strongly) convex functions and a deterministic gradient, NAG achieves a global convergence rate of $O(1/T^2)$ (versus the $O(1/T)$ of gradient descent), with constant proportional to the Lipschitz coefficient of the derivative and the squared Euclidean distance to the solution. While NAG is not typically thought of as a type of momentum, it indeed turns out to be closely related to classical momentum, differing only in the precise update of the velocity vector v, the significance of which we will discuss in the next sub-section. Specifically, as shown in the appendix, the NAG update may be rewritten as:

$$v_{t+1} = \mu v_t - \varepsilon \nabla f(\theta_t + \mu v_t) \quad (3)$$

$$\theta_{t+1} = \theta_t + v_{t+1} \quad (4)$$

While the classical convergence theories for both methods rely on noiseless gradient estimates (i.e., not stochastic), with some care in practice they are both applicable to the stochastic setting. However, the theory predicts that any advantages in terms of asymptotic local rate of convergence will be lost (Orr, 1996; Wiegerinck et al., 1999), a result also confirmed in experiments (LeCun et al., 1998). For these reasons, interest in momentum methods diminished after they had received substantial attention in the 90's. And because of this apparent incompatibility with stochastic optimization, some authors even discourage using momentum or downplay its potential advantages (LeCun et al., 1998).

However, while local convergence is all that matters in terms of asymptotic convergence rates (and on certain very simple/shallow neural network optimization problems it may even dominate the total learning time), in practice, the "transient phase" of convergence (Darken & Moody, 1993), which occurs before fine local convergence sets in, seems to matter a lot more for optimizing deep neural networks. In this transient phase of learning, directions of reduction in the objective tend to persist across many successive gradient estimates and are not completely swamped by noise.

Although the transient phase of learning is most noticeable in training deep learning models, it is still noticeable in convex objectives. The convergence rate of stochastic gradient descent on smooth convex functions is given by $O(L/T + \sigma/\sqrt{T})$, where σ is the variance in the gradient estimate and L is the Lipshits coefficient of ∇f. In contrast, the convergence rate of an accelerated gradient method of Lan (2010) (which is related to but different from NAG, in that it combines Nesterov style momentum with dual averaging) is $O(L/T^2 + \sigma/\sqrt{T})$. Thus, for convex objectives, momentum-based methods will outperform SGD in the early or transient stages of the optimization where L/T is the dominant term. However, the two methods will be equally effective during the final stages of the optimization where $\sigma/\sqrt{T}$ is the dominant term (i.e., when the optimization problem resembles an estimation one).

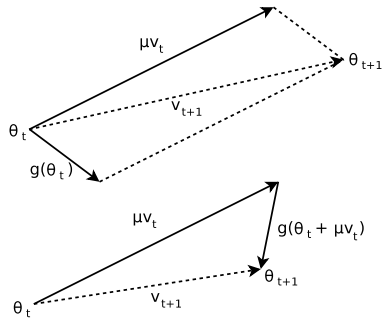

Figure 1. **(Top)** Classical Momentum **(Bottom)** Nesterov Accelerated Gradient

2.1. The Relationship between CM and NAG

From Eqs. 1-4 we see that both CM and NAG compute the new velocity by applying a gradient-based correction to the previous velocity vector (which is decayed), and then add the velocity to θ_t. But while CM computes the gradient update from the current position θ_t, NAG first performs a partial update to θ_t, computing $\theta_t + \mu v_t$, which is similar to θ_{t+1}, but missing the as yet unknown correction. This benign-looking difference seems to allow NAG to change v in a quicker and more responsive way, letting it behave more stably than CM in many situations, especially for higher values of μ.

Indeed, consider the situation where the addition of μv_t results in an immediate undesirable increase in the objective f. The gradient correction to the velocity v_t is computed at position $\theta_t + \mu v_t$ and if μv_t is indeed a poor update, then $\nabla f(\theta_t + \mu v_t)$ will point back towards θ_t more strongly than $\nabla f(\theta_t)$ does, thus providing a larger and more timely correction to v_t than CM. See fig. 1 for a diagram which illustrates this phenomenon geometrically. While each iteration of NAG may only be slightly more effective than CM at correcting a large and inappropriate velocity, this difference in effectiveness may compound as the algorithms iterate. To demonstrate this compounding, we applied both NAG and CM to a two-dimensional oblong quadratic objective, both with the same momentum and learning rate constants (see fig. 2 in the appendix). While the optimization path taken by CM exhibits large oscillations along the high-curvature vertical direction, NAG is able to avoid these oscillations almost entirely, confirming the intuition that it is much more effective than CM at decelerating over the course of multiple iterations, thus making NAG more tolerant of large values of μ compared to CM.

In order to make these intuitions more rigorous and

help quantify precisely the way in which CM and NAG differ, we analyzed the behavior of each method when applied to a positive definite quadratic objective $q(x) = x^\top Ax/2 + b^\top x$. We can think of CM and NAG as operating independently over the different eigendirections of A. NAG operates along any one of these directions equivalently to CM, except with an effective value of μ that is given by $\mu(1 - \lambda\varepsilon)$, where λ is the associated eigenvalue/curvature.

The first step of this argument is to reparameterize $q(x)$ in terms of the coefficients of x under the basis of eigenvectors of A. Note that since $A = U^\top DU$ for a diagonal D and orthonormal U (as A is symmetric), we can reparameterize $q(x)$ by the matrix transform U and optimize $y = Ux$ using the objective $p(y) \equiv q(x) = q(U^\top y) = y^\top U \left(U^\top DU\right) U^\top y/2 + b^\top U^\top y = y^\top Dy/2 + c^\top y$, where $c = Ub$. We can further rewrite p as $p(y) = \sum_{i=1}^n [p]_i([y]_i)$, where $[p]_i(t) = \lambda_i t^2/2 + [c]_i t$ and $\lambda_i > 0$ are the diagonal entries of D (and thus the eigenvalues of A) and correspond to the curvature along the associated eigenvector directions. As shown in the appendix (Proposition 6.1), both CM and NAG, being first-order methods, are "invariant" to these kinds of reparameterizations by orthonormal transformations such as U. Thus when analyzing the behavior of either algorithm applied to $q(x)$, we can instead apply them to $p(y)$, and transform the resulting sequence of iterates back to the default parameterization (via multiplication by $U^{-1} = U^\top$).

Theorem 2.1. *Let $p(y) = \sum_{i=1}^n [p]_i([y]_i)$ such that $[p]_i(t) = \lambda_i t^2/2 + c_i t$. Let ε be arbitrary and fixed. Denote by $CM_{\mathrm{x}}(\mu, p, y, v)$ and $CM_{\mathrm{v}}(\mu, p, y, v)$ the parameter vector and the velocity vector respectively, obtained by applying one step of CM (i.e., Eq. 1 and then Eq. 2) to the function p at point y, with velocity v, momentum coefficient μ, and learning rate ε. Define NAG_{x} and NAG_{v} analogously. Then the following holds for* $\mathrm{z} \in \{\mathrm{x}, \mathrm{v}\}$*:*

$$CM_{\mathrm{z}}(\mu, p, y, v) = \begin{bmatrix} CM_{\mathrm{z}}(\mu, [p]_1, [y]_1, [v]_1) \\ \vdots \\ CM_{\mathrm{z}}(\mu, [p]_n, [y]_n, [v]_n) \end{bmatrix}$$

$$NAG_{\mathrm{z}}(\mu, p, y, v) = \begin{bmatrix} CM_{\mathrm{z}}(\mu(1 - \lambda_1\varepsilon), [p]_1, [y]_1, [v]_1) \\ \vdots \\ CM_{\mathrm{z}}(\mu(1 - \lambda_n\varepsilon), [p]_n, [y]_n, [v]_n) \end{bmatrix}$$

Proof. See the appendix. □

The theorem has several implications. First, CM and NAG become equivalent when ε is small (when $\varepsilon\lambda \ll 1$ for every eigenvalue λ of A), so NAG and CM are distinct only when ε is reasonably large. When ε is relatively large, NAG uses smaller effective momentum for the high-curvature eigen-directions, which prevents oscillations (or divergence) and thus allows the use of a larger μ than is possible with CM for a given ε.

3. Deep Autoencoders

The aim of our experiments is three-fold. First, to investigate the attainable performance of stochastic momentum methods on deep autoencoders starting from well-designed random initializations; second, to explore the importance and effect of the schedule for the momentum parameter μ assuming an optimal fixed choice of the learning rate ε; and third, to compare the performance of NAG versus CM.

For our experiments with feed-forward nets, we focused on training the three deep autoencoder problems described in Hinton & Salakhutdinov (2006) (see sec. A.2 for details). The task of the neural network autoencoder is to reconstruct its own input subject to the constraint that one of its hidden layers is of low-dimension. This "bottleneck" layer acts as a low-dimensional code for the original input, similar to other dimensionality reduction techniques like Principle Component Analysis (PCA). These autoencoders are some of the deepest neural networks with published results, ranging between 7 and 11 layers, and have become a standard benchmarking problem (e.g., Martens, 2010; Glorot & Bengio, 2010; Chapelle & Erhan, 2011; Raiko et al., 2011). See the appendix for more details.

Because the focus of this study is on optimization, we only report training errors in our experiments. Test error depends strongly on the amount of overfitting in these problems, which in turn depends on the type and amount of regularization used during training. While regularization is an issue of vital importance when designing systems of practical utility, it is outside the scope of our discussion. And while it could be objected that the gains achieved using better optimization methods are only due to more exact fitting of the training set in a manner that does not generalize, this is simply not the case in these problems, where undertrained solutions are known to perform poorly on both the training and test sets (underfitting).

The networks we trained used the standard sigmoid nonlinearity and were initialized using the "sparse initialization" technique (SI) of Martens (2010) that is described in sec. 3.1. Each trial consists of 750,000 parameter updates on minibatches of size 200. No regularization is used. The schedule for μ was given by the following formula:

$$\mu_t = \min(1 - 2^{-1-\log_2(\lfloor t/250 \rfloor + 1)}, \mu_{\max}) \qquad (5)$$

where $\mu_{\max}$ was chosen from $\{0.999, 0.995, 0.99, 0.9, 0\}$. This schedule was motivated by Nesterov (1983) who advocates using what amounts to $\mu_t = 1 - 3/(t+5)$ after some manipulation

task	$0_{(\text{SGD})}$	0.9N	0.99N	0.995N	0.999N	0.9M	0.99M	0.995M	0.999M	SGD_C	$\text{HF}^\dagger$	HF^*
Curves	0.48	0.16	0.096	0.091	**0.074**	0.15	0.10	0.10	0.10	0.16	0.058	0.11
Mnist	2.1	1.0	**0.73**	0.75	0.80	1.0	0.77	0.84	0.90	0.9	0.69	1.40
Faces	36.4	14.2	8.5	7.8	**7.7**	15.3	8.7	8.3	9.3	NA	7.5	12.0

Table 1. The table reports the squared errors on the problems for each combination of $\mu_{\max}$ and a momentum type (NAG, CM). When $\mu_{\max}$ is 0 the choice of NAG vs CM is of no consequence so the training errors are presented in a single column. For each choice of $\mu_{\max}$, the highest-performing learning rate is used. The column SGD_C lists the results of Chapelle & Erhan (2011) who used 1.7M SGD steps and tanh networks. The column $HF^\dagger$ lists the results of HF without L2 regularization, as described in sec. 5; and the column HF^* lists the results of Martens (2010).

problem	before	after
Curves	0.096	0.074
Mnist	1.20	0.73
Faces	10.83	7.7

Table 2. The effect of low-momentum finetuning for NAG. The table shows the training squared errors before and after the momentum coefficient is reduced. During the primary ("transient") phase of learning we used the optimal momentum and learning rates.

(see appendix), and by Nesterov (2003) who advocates a constant μ_t that depends on (essentially) the condition number. The constant μ_t achieves exponential convergence on strongly convex functions, while the $1-3/(t+5)$ schedule is appropriate when the function is not strongly convex. The schedule of Eq. 5 blends these proposals. For each choice of $\mu_{\max}$, we report the learning rate that achieved the best training error. Given the schedule for μ, the learning rate ε was chosen from $\{0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001\}$ in order to achieve the lowest final error training error after our fixed number of updates.

Table 1 summarizes the results of these experiments. It shows that NAG achieves the lowest published results on this set of problems, including those of Martens (2010). It also shows that larger values of $\mu_{\max}$ tend to achieve better performance and that NAG usually outperforms CM, especially when $\mu_{\max}$ is 0.995 and 0.999. Most surprising and importantly, the results demonstrate that NAG can achieve results that are comparable with some of the best HF results for training deep autoencoders. Note that the previously published results on HF used L2 regularization, so they cannot be directly compared. However, the table also includes experiments we performed with an improved version of HF (see sec. 2.1) where weight decay was removed towards the end of training.

We found it beneficial to reduce μ to 0.9 (unless μ is 0, in which case it is unchanged) during the final 1000 parameter updates of the optimization without reducing the learning rate, as shown in Table 2. It appears that reducing the momentum coefficient allows for finer convergence to take place whereas otherwise the overly aggressive nature of CM or NAG would prevent this. This phase shift between optimization that favors fast accelerated motion along the error surface (the "transient phase") followed by more careful optimization-as-estimation phase seems consistent with the picture presented by Darken & Moody (1993). However, while asymptotically it is the second phase which must eventually dominate computation time, in practice it seems that for deeper networks in particular, the first phase dominates overall computation time as long as the second phase is cut off before the remaining potential gains become either insignificant or entirely dominated by overfitting (or both).

It may be tempting then to use lower values of μ from the outset, or to reduce it immediately when progress in reducing the error appears to slow down. However, in our experiments we found that doing this was detrimental in terms of the final errors we could achieve, and that despite appearing to not make much progress, or even becoming significantly non-monotonic, the optimizers were doing something apparently useful over these extended periods of time at higher values of μ.

A speculative explanation as to why we see this behavior is as follows. While a large value of μ allows the momentum methods to make useful progress along slowly-changing directions of low-curvature, this may not immediately result in a significant reduction in error, due to the failure of these methods to converge in the more turbulent high-curvature directions (which is especially hard when μ is large). Nevertheless, this progress in low-curvature directions takes the optimizers to new regions of the parameter space that are characterized by closer proximity to the optimum (in the case of a convex objective), or just higher-quality local minimia (in the case of non-convex optimization). Thus, while it is important to adopt a more careful scheme that allows fine convergence to take place along the high-curvature directions, this must be done with care. Reducing μ and moving to this fine convergence regime too early may make it difficult for the optimization to make significant progress along the low-curvature directions, since without the benefit of momentum-based acceleration, first-order methods are notoriously bad at this (which is what motivated the use of second-order methods like HF for deep learning).

SI scale multiplier	0.25	0.5	1	2	4
error	16	16	0.074	0.083	0.35

Table 3. The table reports the training squared error that is attained by changing the scale of the initialization.

3.1. Random Initializations

The results in the previous section were obtained with standard logistic sigmoid neural networks that were initialized with the sparse initialization technique (SI) described in Martens (2010). In this scheme, each random unit is connected to 15 randomly chosen units in the previous layer, whose weights are drawn from a unit Gaussian, and the biases are set to zero. The intuitive justification is that the total amount of input to each unit will not depend on the size of the previous layer and hence they will not as easily saturate. Meanwhile, because the inputs to each unit are not all randomly weighted blends of the outputs of many 100s or 1000s of units in the previous layer, they will tend to be qualitatively more "diverse" in their response to inputs. When using tanh units, we transform the weights to simulate sigmoid units by setting the biases to 0.5 and rescaling the weights by 0.25.

We investigated the performance of the optimization as a function of the scale constant used in SI (which defaults to 1 for sigmoid units). We found that SI works reasonably well if it is rescaled by a factor of 2, but leads to noticeable (but not severe) slow down when scaled by a factor of 3. When we used the factor 1/2 or 5 we did not achieve sensible results.

4. Recurrent Neural Networks

Echo-State Networks (ESNs) is a family of RNNs with an unusually simple training method: their hidden-to-output connections are learned from data, but their recurrent connections are fixed to a random draw from a specific distribution and are not learned. Despite their simplicity, ESNs with many hidden units (or with units with explicit temporal integration, like the LSTM) have achieved high performance on tasks with long range dependencies (**?**). In this section, we investigate the effectiveness of momentum-based methods with ESN-inspired initialization at training RNNs with conventional size and standard (i.e., non-integrating) neurons. We find that momentum-accelerated SGD can successfully train such RNNs on various artificial datasets exhibiting considerable long-range temporal dependencies. This is unexpected because RNNs were believed to be almost impossible to successfully train on such datasets with first-order methods, due to various difficulties such as vanishing/exploding gradients (Bengio et al., 1994). While we found that the use of momentum significantly improved performance and robustness, we obtained nontrivial results even with standard SGD, provided that the learning rate was set low enough.

connection type	sparsity	scale
in-to-hid (add,mul)	dense	$0.001 \cdot N(0,1)$
in-to-hid (mem)	dense	$0.1 \cdot N(0,1)$
hid-to-hid	15 fan-in	spectral radius of 1.1
hid-to-out	dense	$0.1 \cdot N(0,1)$
hid-bias	dense	0
out-bias	dense	average of outputs

Table 4. The RNN initialization used in the experiments. The scale of the vis-hid connections is problem dependent.

Each task involved optimizing the parameters of a randomly initialized RNN with 100 standard tanh hidden units (the same model used by Martens & Sutskever (2011)). The tasks were designed by Hochreiter & Schmidhuber (1997), and are referred to as training "problems". See sec. A.3 of the appendix for details.

4.1. ESN-based Initialization

As argued by Jaeger & Haas (2004), the spectral radius of the hidden-to-hidden matrix has a profound effect on the dynamics of the RNN's hidden state (with a tanh nonlinearity). When it is smaller than 1, the dynamics will have a tendency to quickly "forget" whatever input signal they may have been exposed to. When it is much larger than 1, the dynamics become oscillatory and chaotic, allowing it to generate responses that are varied for different input histories. While this allows information to be retained over many time steps, it can also lead to severe exploding gradients that make gradient-based learning much more difficult. However, when the spectral radius is only slightly greater than 1, the dynamics remain oscillatory and chaotic while the gradient are no longer exploding (and if they do explode, then only "slightly so"), so learning may be possible with a spectral radius of this order. This suggests that a spectral radius of around 1.1 may be effective.

To achieve robust results, we also found it is essential to carefully set the initial scale of the input-to-hidden connections. When training RNNs to solve those tasks that possess many random and irrelevant distractor inputs, we found that having the scale of these connections set too high at the start led to relevant information in the hidden state being too easily "overwritten" by the many irrelevant signals, which ultimately led the optimizer to converge towards an extremely poor local minimum where useful information was never relayed over long distances. Conversely, we found that if this scale was set too low, it led to significantly slower learning. Having experimented with multiple scales we found that a Gaussian draw with a standard deviation of 0.001 achieved a good balance between these concerns. However, unlike the value of 1.1 for the spectral radius of the dynamics matrix, which worked well on all tasks, we found that good choices for initial scale of the input-to-hidden weights depended a lot on the particular characteristics of the particular task (such

as its dimensionality or the input variance). Indeed, for tasks that do not have many irrelevant inputs, a larger scale of the input-to-hidden weights (namely, 0.1) worked better, because the aforementioned disadvantage of large input-to-hidden weights does not apply. See table 4 for a summary of the initializations used in the experiments. Finally, we found centering (mean subtraction) of both the inputs and the outputs to be important to reliably solve all of the training problems. See the appendix for more details.

4.2. Experimental Results

We conducted experiments to determine the efficacy of our initializations, the effect of momentum, and to compare NAG with CM. Every learning trial used the aforementioned initialization, 50,000 parameter updates and on minibatches of 100 sequences, and the following schedule for the momentum coefficient μ: $\mu = 0.9$ for the first 1000 parameter, after which $\mu = \mu_0$, where μ_0 can take the following values $\{0, 0.9, 0.98, 0.995\}$. For each μ_0, we use the empirically best learning rate chosen from $\{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$.

The results are presented in Table 5, which are the average loss over 4 different random seeds. Instead of reporting the loss being minimized (which is the squared error or cross entropy), we use a more interpretable zero-one loss, as is standard practice with these problems. For the bit memorization, we report the fraction of timesteps that are computed incorrectly. And for the addition and the multiplication problems, we report the fraction of cases where the RNN the error in the final output prediction exceeded 0.04.

Our results show that despite the considerable long-range dependencies present in training data for these problems, RNNs can be successfully and robustly trained to solve them, through the use of the initialization discussed in sec. 4.1, momentum of the NAG type, a large μ_0, and a particularly small learning rate (as compared with feedforward networks). Our results also suggest that with larger values of μ_0 achieve better results with NAG but not with CM, possibly due to NAG's tolerance of larger μ_0's (as discussed in sec. 2).

Although we were able to achieve surprisingly good training performance on these problems using a sufficiently strong momentum, the results of Martens & Sutskever (2011) appear to be moderately better and more robust. They achieved lower error rates and their initialization was chosen with less care, although the initializations are in many ways similar to ours. Notably, Martens & Sutskever (2011) were able to solve these problems without centering, while we had to use centering to solve the multiplication problem (the other problems are already centered). This suggests that the initialization proposed here, together with the method of Martens & Sutskever (2011), could achieve even better performance. But the main achievement of these results is a demonstration of the ability of momentum methods to cope with long-range temporal dependency training tasks to a level which seems sufficient for most practical purposes. Moreover, our approach seems to be more tolerant of smaller minibatches, and is considerably simpler than the particular version of HF proposed in Martens & Sutskever (2011), which used a specialized update damping technique whose benefits seemed mostly limited to training RNNs to solve these kinds of extreme temporal dependency problems.

5. Momentum and HF

Truncated Newton methods, that include the HF method of Martens (2010) as a particular example, work by optimizing a local quadratic model of the objective via the linear conjugate gradient algorithm (CG), which is a first-order method. While HF, like all truncated-Newton methods, takes steps computed using partially converged calls to CG, it is naturally accelerated along at least some directions of lower curvature compared to the gradient. It can even be shown (Martens & Sutskever, 2012) that CG will tend to favor convergence to the exact solution to the quadratic sub-problem first along higher curvature directions (with a bias towards those which are more clustered together in their curvature-scalars/eigenvalues).

While CG accumulates information as it iterates which allows it to be optimal in a much stronger sense than any other first-order method (like NAG), once it is terminated, this information is lost. Thus, standard truncated Newton methods can be thought of as persisting information which accelerates convergence (of the current quadratic) only over the number of iterations CG performs. By contrast, momentum methods persist information that can inform new updates across an arbitrary number of iterations.

One key difference between standard truncated Newton methods and HF is the use of "hot-started" calls to CG, which use as their initial solution the one found at the previous call to CG. While this solution was computed using old gradient and curvature information from a previous point in parameter space and possibly a different set of training data, it may be well-converged along certain eigen-directions of the new quadratic, despite being very poorly converged along others (perhaps worse than the default initial solution of $\vec{0}$). However, to the extent to which the new local quadratic model resembles the old one, and in particular in the more difficult to optimize directions of low-curvature (which will arguably be more likely to persist across nearby locations in parameter space), the previous solution will be a preferable starting point to 0, and may even allow for gradually increasing levels of convergence along certain directions which persist

problem	biases	0	0.9N	0.98N	0.995N	0.9M	0.98M	0.995M
add $T = 80$	0.82	0.39	0.02	0.21	**0.00025**	0.43	0.62	0.036
mul $T = 80$	0.84	0.48	0.36	0.22	**0.0013**	0.029	0.025	0.37
mem-5 $T = 200$	2.5	1.27	1.02	0.96	**0.63**	1.12	1.09	0.92
mem-20 $T = 80$	8.0	5.37	2.77	0.0144	**0.00005**	1.75	0.0017	0.053

Table 5. Each column reports the errors (zero-one losses; sec. 4.2) on different problems for each combination of μ_0 and momentum type (NAG, CM), averaged over 4 different random seeds. The "biases" column lists the error attainable by learning the output biases and ignoring the hidden state. This is the error of an RNN that failed to "establish communication" between its inputs and targets. For each μ_0, we used the fixed learning rate that gave the best performance.

in the local quadratic models across many updates.

The connection between HF and momentum methods can be made more concrete by noticing that a single step of CG is effectively a gradient update taken from the current point, plus the previous update reapplied, just as with NAG, and that if CG terminated after just 1 step, HF becomes equivalent to NAG, except that it uses a special formula based on the curvature matrix for the learning rate instead of a fixed constant. The most effective implementations of HF even employ a "decay" constant (Martens & Sutskever, 2012) which acts analogously to the momentum constant μ. Thus, in this sense, the CG initializations used by HF allow us to view it as a hybrid of NAG and an exact second-order method, with the number of CG iterations used to compute each update effectively acting as a dial between the two extremes.

Inspired by the surprising success of momentum-based methods for deep learning problems, we experimented with making HF behave even more like NAG than it already does. The resulting approach performed surprisingly well (see Table 1). For a more detailed account of these experiments, see sec. A.6 of the appendix.

If viewed on the basis of each CG step (instead of each update to parameters θ), HF can be thought of as a peculiar type of first-order method which approximates the objective as a series of quadratics only so that it can make use of the powerful first-order CG method. So apart from any potential benefit to global convergence from its tendency to prefer certain directions of movement in parameter space over others, perhaps the main theoretical benefit to using HF over a first-order method like NAG is its use of CG, which, while itself a first-order method, is well known to have strongly optimal convergence properties for quadratics, and can take advantage of clustered eigenvalues to accelerate convergence (see Martens & Sutskever (2012) for a detailed account of this well-known phenomenon). However, it is known that in the worst case that CG, when run in batch mode, will converge asymptotically no faster than NAG (also run in batch mode) for certain specially designed quadratics with very evenly distributed eigenvalues/curvatures. Thus it is worth asking whether the quadratics which arise during the optimization of neural networks by HF are such that CG has a distinct advantage in optimizing them over NAG, or if they are closer to the aforementioned worst-case examples. To examine this question we took a quadratic generated during the middle of a typical run of HF on the curves dataset and compared the convergence rate of CG, initialized from zero, to NAG (also initialized from zero). Figure 5 in the appendix presents the results of this experiment. While this experiment indicates some potential advantages to HF, the closeness of the performance of NAG and HF suggests that these results might be explained by the solutions leaving the area of trust in the quadratics before any extra speed kicks in, or more subtly, that the faithfulness of approximation goes down just enough as CG iterates to offset the benefit of the acceleration it provides.

6. Discussion

Martens (2010) and Martens & Sutskever (2011) demonstrated the effectiveness of the HF method as a tool for performing optimizations for which previous attempts to apply simpler first-order methods had failed. While some recent work (Chapelle & Erhan, 2011; Glorot & Bengio, 2010) suggested that first-order methods can actually achieve some success on these kinds of problems when used in conjunction with good initializations, their results still fell short of those reported for HF. In this paper we have completed this picture and demonstrated conclusively that a large part of the remaining performance gap that is not addressed by using a well-designed random initialization is in fact addressed by careful use of momentum-based acceleration (possibly of the Nesterov type). We showed that careful attention must be paid to the momentum constant μ, as predicted by the theory for local and convex optimization.

Momentum-accelerated SGD, despite being a first-order approach, is capable of accelerating directions of low-curvature just like an approximate Newton method such as HF. Our experiments support the idea that this is important, as we observed that the use of stronger momentum (as determined by μ) had a dramatic effect on optimization performance, particularly for the RNNs. Moreover, we showed that HF can be viewed as a first-order method, and as a generalization of NAG in particular, and that it already derives some of its benefits through a momentum-like mechanism.

References

Bengio, Y., Simard, P., and Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5:157–166, 1994.

Bengio, Y., Lamblin, P, Popovici, D., and Larochelle, H. Greedy layer-wise training of deep networks. In *In NIPS*. MIT Press, 2007.

Bottou, L. and LeCun, Y. Large scale online learning. In *Advances in Neural Information Processing Systems 16: Proceedings of the 2003 Conference*, volume 16, pp. 217. MIT Press, 2004.

Chapelle, O. and Erhan, D. Improved Preconditioner for Hessian Free Optimization. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

Cotter, A., Shamir, O., Srebro, N., and Sridharan, K. Better mini-batch algorithms via accelerated gradient methods. *arXiv preprint arXiv:1106.4574*, 2011.

Dahl, G.E., Yu, D., Deng, L., and Acero, A. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):30–42, 2012.

Darken, C. and Moody, J. Towards faster stochastic gradient search. *Advances in neural information processing systems*, pp. 1009–1009, 1993.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of AISTATS 2010*, volume 9, pp. 249–256, may 2010.

Graves, A. Sequence transduction with recurrent neural networks. *arXiv preprint arXiv:1211.3711*, 2012.

Hinton, G and Salakhutdinov, R. Reducing the dimensionality of data with neural networks. *Science*, 313:504–507, 2006.

Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T., et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, 2012.

Hinton, G.E., Osindero, S., and Teh, Y.W. A fast learning algorithm for deep belief nets. *Neural computation*, 18 (7):1527–1554, 2006.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Jaeger, H. personal communication, 2012.

Jaeger, H. and Haas, H. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304:78–80, 2004.

Krizhevsky, A., Sutskever, I., and Hinton, G. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pp. 1106–1114, 2012.

Lan, G. An optimal method for stochastic composite optimization. *Mathematical Programming*, pp. 1–33, 2010.

LeCun, Y., Bottou, L., Orr, G., and Müller, K. Efficient backprop. *Neural networks: Tricks of the trade*, pp. 546–546, 1998.

Martens, J. Deep learning via Hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, 2010.

Martens, J. and Sutskever, I. Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, pp. 1033–1040, 2011.

Martens, J. and Sutskever, I. Training deep and recurrent networks with hessian-free optimization. *Neural Networks: Tricks of the Trade*, pp. 479–535, 2012.

Mikolov, Tomáš, Sutskever, Ilya, Deoras, Anoop, Le, Hai-Son, Kombrink, Stefan, and Cernocky, J. Subword language modeling with neural networks. *preprint (http://www. fit. vutbr. cz/imikolov/rnnlm/char. pdf)*, 2012.

Mohamed, A., Dahl, G.E., and Hinton, G. Acoustic modeling using deep belief networks. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):14 –22, Jan. 2012.

Nesterov, Y. A method of solving a convex programming problem with convergence rate O(1/sqr(k)). *Soviet Mathematics Doklady*, 27:372–376, 1983.

Nesterov, Y. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer, 2003.

Orr, G.B. Dynamics and algorithms for stochastic search. 1996.

Polyak, B.T. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

Raiko, Tapani, Valpola, Harri, and LeCun, Yann. Deep learning made easier by linear transformations in perceptrons. In *NIPS 2011 Workshop on Deep Learning and Unsupervised Feature Learning, Sierra Nevada, Spain*, 2011.

Sutskever, I., Martens, J., and Hinton, G. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning*, ICML '11, pp. 1017–1024, June 2011.

Wiegerinck, W., Komoda, A., and Heskes, T. Stochastic dynamics of learning with momentum in neural networks. *Journal of Physics A: Mathematical and General*, 27(13):4425, 1999.

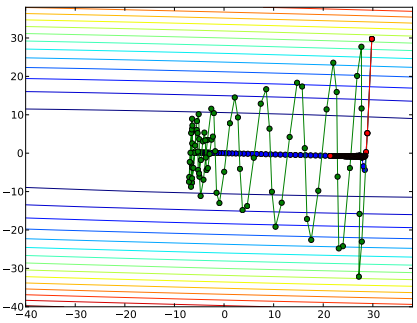

Figure 2. The trajectories of CM, NAG, and SGD are shown. Although the value of the momentum is identical for both experiments, CM exhibits oscillations along the high-curvature directions, while NAG exhibits no such oscillations. The global minimizer of the objective is at (0,0). The red curve shows 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.

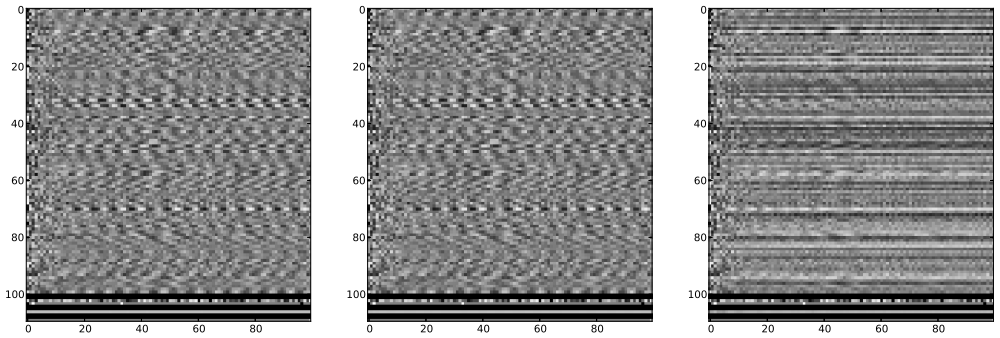

Figure 3. In each figure, the vertical axis is time, and the horizontal axis are units. The leftmost figure shows the hidden state sequences of an RNN on the addition problem with the aforementioned parameter initialization. Note the "gentle" oscillations of the hidden states. The middle figure shows 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.

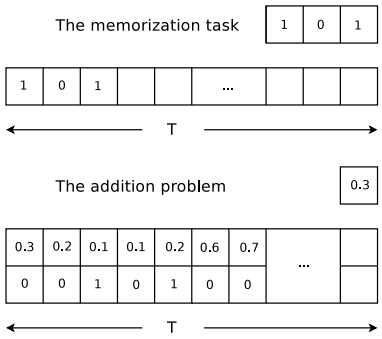

Figure 4. The memorization and the addition problems. The goal of the memorization problem is to output the input bits in the correct order. The goal of the addition and the multiplication problem is to output the sum (or the product) of the two marked inputs.

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

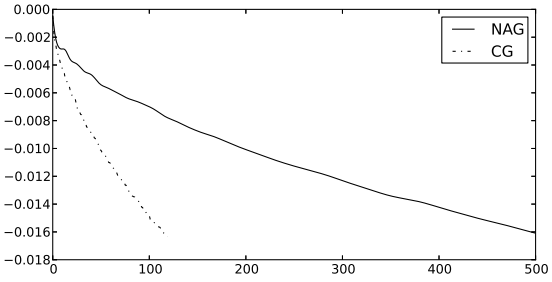

Figure 5. Comparison of NAG and CG on a damped quadratic objective from the middle of HF. The plot shows that while CG is considerably more effective than NAG at opitimizing the kinds of quadratic approximation that occur during neural network learning, although the difference is not dramatic. Note that 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.