Learning Recurrent Neural Networks with Hessian-Free Optimization

James Martens JMARTENS@CS.TORONTO.EDU
Ilya Sutskever ILYA@CS.UTORONTO.CA
University of Toronto, Canada

Abstract

In this work we resolve the long-outstanding problem of how to effectively train recurrent neural networks (RNNs) on complex and difficult sequence modeling problems which may contain long-term data dependencies. Utilizing recent advances in the Hessian-free optimization approach (Martens, 2010), together with a novel damping scheme, we successfully train RNNs on two sets of challenging problems. First, a collection of pathological synthetic datasets which are known to be impossible for standard optimization approaches (due to their extremely long-term dependencies), and second, on three natural and highly complex real-world sequence datasets where we find that our method significantly outperforms the previous state-of-the-art method for training neural sequence models: the Long Short-term Memory approach of Hochreiter and Schmidhuber (1997). Additionally, we offer a new interpretation of the generalized Gauss-Newton matrix of Schraudolph (2002) which is used within the HF approach of Martens.

1. Introduction

A Recurrent Neural Network (RNN) is a neural network that operates in time. At each timestep, it accepts an input vector, updates its (possibly high-dimensional) hidden state via non-linear activation functions, and uses it to make a prediction of its output. RNNs form a rich model class because their hidden state can store information as high-dimensional distributed representations (as opposed to a Hidden Markov Model, whose hidden state is essentially $\log n$-dimensional) and their nonlinear dynamics can implement rich and powerful computations, allowing the RNN to perform modeling and prediction tasks for sequences with highly complex structure.

Appearing in *Proceedings of the 28^{th} International Conference on Machine Learning*, Bellevue, WA, USA, 2011.

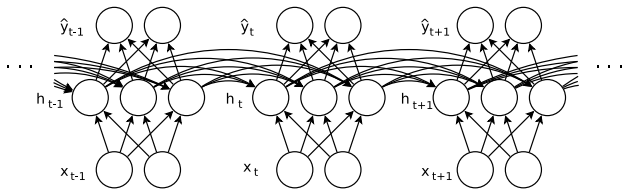

Figure 1. The architecture of a recurrent neural network.

Gradient-based training of RNNs might appear straightforward because, unlike many rich probabilistic sequence models (Murphy, 2002), the exact gradients can be cheaply computed by the Backpropagation Through Time (BPTT) algorithm (Rumelhart et al., 1986). Unfortunately, gradient descent and other 1st-order methods completely fail to properly train RNNs on large families of seemingly simple yet pathological synthetic problems that separate a target output and from its relevant input by many time steps (Bengio et al., 1994; Hochreiter and Schmidhuber, 1997). In fact, 1st-order approaches struggle even when the separation is only 10 timesteps (Bengio et al., 1994). An unfortunate consequence of these failures is that these highly expressive and potentially very powerful time-series models are seldom used in practice.

The extreme difficulty associated with training RNNs is likely due to the highly volatile relationship between the parameters and the hidden states. One way that this volatility manifests itself, which has a direct impact on the performance of gradient-descent, is in the so-called "vanishing/exploding gradients" phenomenon (Bengio et al., 1994; Hochreiter, 1991), where the error-signals exhibit exponential decay/growth as they are back-propagated through time. In the case of decay, this leads to the long-term error signals being effectively lost as they are overwhelmed by un-decayed short-term signals, and in the case of exponential growth there is the opposite problem that the short-term error signals are overwhelmed by the long-term ones.

During the 90's there was intensive research by the machine learning community into identifying the source of difficultly in training RNNs as well as proposing methods to address it. However, none of these methods became widely adopted, and an analysis by Hochreiter and Schmidhuber (1996) showed that they were often no better than random guessing. In an attempt to sidestep the difficulty of training RNNs on problems exhibiting long-

term dependencies, Hochreiter and Schmidhuber (1997) proposed a modified architecture called the Long Short-term Memory (LSTM) and successfully applied it to speech and handwritten text recognition (Graves and Schmidhuber, 2009; 2005) and robotic control (Mayer et al., 2007). The LSTM consists of a standard RNN augmented with "memory-units" which specialize in transmitting long-term information, along with a set of "gating" units which allow the memory units to selectively interact with the usual RNN hidden state. Because the memory units are forced to have fixed linear dynamics with a self-connection of value 1, the error signal neither decays nor explodes as it is backpropagated through them. However, it is not clear if the approach of training with gradient descent and enabling long-term memorization with the specialized LSTM architecture is harnessing the true power of recurrent neural computation. In particular, gradient descent may be deficient for RNN learning in ways that are not compensated for by using LSTMs. Another recent attempt to resolve the problem of RNN training which has received attention is the Echo-State-Network (ESN) of Jaeger and Haas (2004), which gives up on learning the problematic hidden-to-hidden weights altogether in favor of using fixed sparse connections which are generated randomly. However, since this approach cannot learn new non-linear dynamics, instead relying on those present in the random "reservoir" which is effectively created by the random initialization, its power is clearly limited.

Recently, a special variant of the Hessian-Free (HF) optimization approach (aka truncated-Newton or Newton-CG) was successfully applied to learning deep multilayered neural networks from random initializations (Martens, 2010), an optimization problem for which gradient descent and even quasi-Newton methods like L-BFGS have never been demonstrated to be effective. Martens examines the problem of learning deep auto-encoders and is able to obtain the best-known results for these problems, significantly surpassing the benchmark set by the seminal deep learning work of Hinton and Salakhutdinov (2006).

Inspired by the success of the HF approach on deep neural networks, in this paper we revisit and resolve the long-standing open problem of RNN training. In particular, our results show that the HF optimization approach of Martens, augmented with a novel "structural-damping" which we develop, can effectively train RNNs on the aforementioned pathological long-term dependency problems (adapted directly from Hochreiter and Schmidhuber (1997)), thus overcoming the main objection made against using RNNs. From there we go on to address the question of whether these advances generalize to real-world sequence modeling problems by considering both a high-dimensional motion video prediction task, a MIDI-music modeling task, and a speech modelling task. We find that RNNs trained using our method are highly effective on these tasks and significantly outperform similarly sized LSTMs.

Other contributions we make include the development of the aforementioned structural damping scheme which, as we demonstrate through experiments, can significantly improve robustness of the HF optimizer in the setting of RNN training, and a new interpretation of the generalized Gauss-Newton matrix of Schraudolph (2002) which forms a key component of the HF approach of Martens.

2. Recurrent Neural Networks

We now formally define the standard RNN (Rumelhart et al., 1986) which forms the focus of this work. Given a sequence of inputs $x_1, x_2, \ldots, x_T$, each in $\mathbb{R}^n$, the network computes a sequence of hidden states $h_1, h_2, \ldots, h_T$, each in $\mathbb{R}^m$, and a sequence of predictions $\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_T$, each in $\mathbb{R}^k$, by iterating the equations

$$t_i = W_{hx} x_i + W_{hh} h_{i-1} + b_h \quad (1)$$

$$h_i = e(t_i) \quad (2)$$

$$s_i = W_{yh} h_i + b_y \quad (3)$$

$$\hat{y}_i = g(s_i) \quad (4)$$

where W_{yh}, W_{hx}, W_{hh} are the weight matrices and b_h, b_y are the biases, $t_1, t_2, \ldots, t_T$, each in $\mathbb{R}^k$, and $s_1, s_2, \ldots, s_T$, each in $\mathbb{R}^k$, are the inputs to the hidden and output units (resp.) and e and g are pre-defined vector valued functions which are typically non-linear and applied coordinate-wise (although the only formal requirement is that they have a well-defined Jacobian). The RNN also has a special initial bias $b_h^{\text{init}} \in \mathbb{R}^m$ which replaces the formally undefined expression $W_{hh} h_0$ at time $i = 1$. For a subscripted variable z_i, the variable without a subscript, z, will refer to all of the z_i's from $i = 1$ to T concatenated into a large vector, and similarly will use θ to refer to all of the parameters as one large vector.

The objective function associated with RNNs for a single training pair (x, y) is defined as $f(\theta) = L(\hat{y}; y)$, where L is a distance function[1] which measures the deviation of the predictions $\hat{y}$ from the target outputs y. Examples of L include the squared error $\sum_i \|\hat{y}_i - y_i\|^2/2$ and the cross-entropy error $-\sum_i \sum_j y_{ij} \log(\hat{y}_{ij}) + (1 - y_{ij}) \log(1 - \hat{y}_{ij})$. The overall objective function for the whole training set is simply given by the average of the individual objectives associated with each training example.

3. Hessian-Free Optimization

Hessian-Free optimization is concerned with the minimization of an objective $f : \mathbb{R}^N \to \mathbb{R}$, with respect to N-dimensional input vector θ. Its operation is viewed as the iterative minimization of simpler sub-objectives based on local approximations to $f(\theta)$. Specifically, given a parameter setting θ_n, the next one, θ_{n+1}, is found by partially optimizing the sub-objective

$$q_{\theta_n}(\theta) \equiv M_{\theta_n}(\theta) + \lambda R_{\theta_n}(\theta), \quad (5)$$

[1] Not necessarily symmetric or sub-additive.

In this equation, $M_{\theta_n}(\theta)$ is a θ_n-dependent "local" quadratic approximation to $f(\theta)$ given by

$$M_{\theta_n}(\theta) = f(\theta_n) + f'(\theta_n)^\top \delta_n + \delta_n^\top \mathrm{B} \delta_n / 2, \quad (6)$$

where B is an approximation to the curvature of f, the term δ_n is given by $\delta_n = \theta - \theta_n$, and $R_{\theta_n}(\theta)$ is a damping function that penalizes the solution according to the difference between θ and θ_n, thus encouraging θ to remain close to θ_n. The use of a damping function is generally necessary because the accuracy of the local approximation $M_{\theta_n}(\theta)$ degrades as θ moves further from θ_n.

In standard Hessian-free optimization (or 'truncated-Newton', as it is also known), M_{θ_n} is chosen to be the same quadratic as is optimized in Newton's method: the full 2nd-order Taylor series approximation to f, which is obtained by taking $\mathrm{B} = f''(\theta_n)$ in eq. 6, and the damping function $R_{\theta_n}(\theta)$ is chosen to be $\|\delta_n\|^2/2$. Unlike with quasi-Newton approaches like L-BFGS there is no low-rank or diagonal approximation involved, and as a consequence, fully optimizing the sub-objective q_{θ_n} can require a matrix inversion or some similarly expensive operation. The HF approach circumvents this problem by performing a much cheaper partial optimization of q_{θ_n} using the linear conjugate gradient algorithm (CG). By utilizing the complete curvature information given by B, CG running within HF can take large steps in directions of low reduction and curvature which are effectively invisible to gradient descent. It turns out that HF's ability to find and strongly pursue these directions is the critical property which allows it to successfully optimize deep neural networks (Martens, 2010).

Although the HF approach has been known and studied for decades within the optimization literature, the shortcomings of existing versions of the approach made them impractical or even completely ineffective for neural net training (Martens, 2010). The version of HF developed by Martens makes a series of important modifications and design choices to the basic approach, which include (among others): using the positive semi-definite Gauss-Newton curvature matrix in place of the possibly indefinite Hessian, using a quadratic damping function for $R_{\theta_n}(\theta) = \|\delta_n\|^2/2$ with the damping parameter λ adjusted by Levenberg-Marquardt style heuristics (Nocedal and Wright, 1999), using a criterion based directly on the value of q_{θ_n} in order to terminate CG (as opposed to the usual residual-error based criterion), and computing the curvature-matrix products $\mathrm{B}v$ using mini-batches (as opposed to the whole training set) and the gradients with much larger mini-batches.

In applying HF to RNNs we deviate from the approach of Martens in two significant ways. First, we use a different damping function for R (developed in section 3.2), which we found improves the overall robustness of the HF algorithm on RNNs by more accurately identifying regions where M is likely to deviate significantly from f. Second, we do not use diagonal preconditioning because we found that it gave no substantive benefit for RNNs[2].

3.1. Multiplication by the Generalized Gauss-Newton Matrix

We now define the aforementioned Gauss-Newton matrix G_f, discuss its properties, and describe the algorithm for computing the curvature matrix-vector product $\mathrm{G}_f v$ (which was first applied to neural networks by Schraudolph (2002), extending the work of Pearlmutter (1994)).

The Gauss-Newton matrix for a single training example (x, y) is given by

$$\mathrm{G}_f \equiv \mathrm{J}_{s,\theta}^\top (L \circ g)'' \mathrm{J}_{s,\theta}\big|_{\theta=\theta_n} \quad (7)$$

where the $\mathrm{J}_{s,\theta}$ is the Jacobian of s w.r.t. θ, and $(L \circ g)''$ is the Hessian of the $L(g(s); y)$ with respect to s. To compute the curvature matrix-vector product $\mathrm{G}_f v$ we first compute $\mathrm{J}_{s,\theta} v = \mathrm{R}s$ using a single forward pass of the "$R\{\}$-method" described in Pearlmutter (1994). Next, we run standard backpropagation through time with the vector $(L \circ g)'' \mathrm{J}_{s,\theta} v$ which effectively accomplishes the multiplication by $\mathrm{J}_{s,\theta}^\top$, giving

$$\mathrm{J}_{s,\theta}^\top ((L \circ g)'' \mathrm{J}_{s,\theta} v)\big|_{\theta=\theta_n} = (\mathrm{J}_{s,\theta}^\top (L \circ g)'' \mathrm{J}_{s,\theta})\big|_{\theta=\theta_n} v = \mathrm{G}_f v$$

If s is precomputed then the total cost of this operation is the same as a standard forward pass and back-propagation through time operation - i.e., it is essentially linear in the number of parameters.

A very important property of the Gauss-Newton matrix is that it is positive semi-definite when $(L \circ g)''$ is, which happens when $L(\hat{y}; y)$ is convex w.r.t. s. Moreover, it can be shown that G_f approaches the Hessian H when the loss $L(\hat{y}(\theta); y)$ approaches 0.

The usual view of the Gauss-Newton matrix is that it is simply a positive semi-definite approximation to the Hessian. There is an alternative view, which is will known in the context of non-linear least-squares problems, which we now extend to encompass Schraudolph's generalized Gauss-Newton formulation. Consider the approximation $\tilde{f}$ of f obtained by "linearizing" the network up to the activations of the output units s, after which the final nonlinearity g and the usual error function are applied. To be precise, we replace $s(\theta)$ with $s(\theta_n) + \mathrm{J}_{s,\theta}|_{\theta=\theta_n} \delta_n$ (recall $\delta_n \equiv \theta - \theta_n$), where $\mathrm{J}_{s,\theta}$ is the Jacobian of s w.r.t. to θ, giving:

$$\tilde{f}_{\theta_n}(\theta) = L\Big(g(\, s(\theta_n) + \mathrm{J}_{s,\theta}|_{\theta=\theta_n} \delta_n \,) \;;\; y \Big) \quad (8)$$

[2]The probable reason for this is that RNNs have no large axis-aligned (i.e. associated to specific parameters) scale differences, since the gradients and curvatures for the various parameters are formed from the summed contributions of both decay and undecayed error signals. In contrast, deep nets exhibit substantial layer-dependent (and thus axis aligned) scale variation due to the fact that all signals which reach a given parameter go through the same number of layers and are thus decayed by roughly the same amount.

Because function $\mathrm{J}_{s,\theta}\delta_n$ is affine in θ, it follows that $\tilde{f}$ is convex when $L \circ g$ is (since it will be the composition of a convex and affine function), and so its Hessian will be positive semi-definite. A sufficient condition for $L \circ g$ being convex is when the output non-linearity function g 'matches' L (see Schraudolph, 2002), as is the case when g is the logistic function and L the cross-entropy error. Note that the choice to linearize up to s is arbitrary in a sense, and we could instead linearize up to $\hat{y}$ (for example), as long as L is convex w.r.t. $\hat{y}$.

In the case of non-linear least-squares, once we linearize the network, $L \circ g$ is simply squared L_2 norm and so $\tilde{f}$ is already quadratic in θ. However, in the more general setting we only require that $L \circ g$ be convex and twice differentiable. To obtain a (convex) quadratic we can simply compute the 2nd-order Taylor series approximation of $\tilde{f}$.

The gradient of $\tilde{f}(\theta)$ is given by

$$\nabla_s L(g(s(\theta_n) + \mathrm{J}_{s,\theta}|_{\theta=\theta_n}\, \delta_n); y)\; \mathrm{J}_{s,\theta}|_{\theta=\theta_n} \qquad (9)$$

The 1st-order term in the quadratic approximation of $\tilde{f}$ centered at θ_n is the gradient of $\tilde{f}$ as computed above, evaluated at $\theta = \theta_n$ (i.e. at $\delta_n = 0$), and is given by

$$\nabla_s L(g(s(\theta_n)); y)\; \mathrm{J}_{s,\theta}|_{\theta=\theta_n}$$

which is equal to the gradient of the exact f evaluated at $\theta = \theta_n$. Thus the 1st-order terms of the quadratic approximations to f and $\hat{f}$ are identical.

Computing the Hessian of $\tilde{f}$ we obtain:

$$\mathrm{J}_{s,\theta}^{\top}\big|_{\theta=\theta_n}\, (L\circ g)''\left(g(s(\theta_n) + \mathrm{J}_{s,\theta}|_{\theta=\theta_n}\, \delta_n); y\right)\, \mathrm{J}_{s,\theta}|_{\theta=\theta_n}$$

The 2nd-order term of the Taylor-series approximation of $\tilde{f}$ is this quantity evaluated at $\theta = \theta_n$. Unlike the first-order term, it is not the same as the corresponding term in the Taylor series of f, but is in fact the Gauss-Newton matrix G_f of f. Thus we showed that the quadratic model of f which uses the Gauss-Newton matrix in place of the Hessian is in fact the Taylor-series approximation to $\tilde{f}$.

Martens (2010) found that using the Gauss-Newton matrix G_f instead of H within the quadratic model for f was highly preferable for several reasons. Firstly, because H is in general indefinite, the sub-objective q_{θ_n} may not even be bounded below when $\mathrm{B} = \mathrm{H} + \lambda\mathrm{I}$. In particular, infinite reduction will be possible along any directions of negative curvature that have a non-zero inner product with the gradient. While this problem can be remedied by choosing λ to be larger in magnitude than the most-negative eigenvalue of H, there is no efficient way to compute this in general, and moreover, using large values of λ can have a highly detrimental effect on the performance of HF as it effectively masks out the important low curvature directions. Secondly, Martens (2010) made the qualitative observation that, even when putting the issues of negative curvature aside, the search directions produced by using G_f in place of H performed much better in practice in the context of neural network learning.

3.2. Structural Damping

While we have found that applying the HF approach of Martens (2010) to RNNs achieves robust performance for *most* of the pathological long-term dependency problems (section 4) without any significant modification, for truly robust performance on all of these problems for 100 timesteps and beyond we found that it was necessary to incorporate an additional idea which we call "structural damping" (named so since it is a damping strategy that makes use of the specific structure of the objective).

In general, 2nd-order optimization algorithms, and HF in particular, perform poorly when the quadratic approximation M becomes highly inaccurate as δ_n approaches the optimum of the quadratic sub-objective q_{θ_n}. Damping, which is critical to the success of HF, encourages the optimal point to be close to $\delta_n = 0$, where the approximation becomes exact but trivial. The basic damping strategy used by Martens is the classic Tikhonov regularization, which penalizes δ_n by $\lambda R(\delta_n)$ for $R(\delta_n) = \frac{1}{2}\|\delta_n\|^2$. This is accomplished by adding $\lambda\mathrm{I}$ to the curvature matrix B.

Our initial experience training RNNs with HF on long-term dependency tasks was that when λ was small there would be a rapid decrease in the accuracy of the quadratic approximation M as δ_n was driven by CG towards the optimum of q_{θ_n}. This would cause the Levenburg-Marquardt heuristics to (rightly) compensate by adjusting λ to be much larger. Unfortunately, we found that such a scenario was associated with poor training outcomes on the more difficult long-term dependency tasks of Hochreiter and Schmidhuber (1997). In particular, we saw that the parameters seemed to approach a bad local minimum where little-to-no long-term information was being propagated through the hidden states.

One way to explain this observation is that for large values of λ any Tikhonov-damped 2nd-order optimization approach (such as HF) will behave similarly to a 1st-order approach[3] and crucial low-curvature directions whose associated curvature is significantly smaller than λ will be effectively masked-out. Martens (2010) found that the Tikhonov damping strategy was very effective for training deep autoencoders presumably because, during any point in the optimization, there was a value of λ which would allow M to remain an accurate approximation of f (at q_{θ_n}'s optimum) without being so high as to reduce the method to mostly 1st-order approach. Unfortunately our experience seems to indicate that is less the case with RNNs.

Our hypothesis is that for certain small changes in θ there can be large and highly non-linear changes in the hidden state sequence h (given that W_{hh} is applied iteratively to

[3] it can be easily shown that as $\lambda \to \infty$, minimizing q_{θ_n} is equivalent to a small step of gradient descent.

potentially hundreds of temporal 'layers') and these will not be accurately approximated by M. These ideas motivate the development of a damping function R that can penalize directions in parameter space, which despite not being large in magnitude can nonetheless lead to large changes in the hidden-state sequence. With such a damping scheme we can penalize said directions without requiring λ to be high as to prevent the strong pursuit of other low-curvature directions whose effect on f are modeled more accurately. To this end we define the "structural damping function" as

$$R_{\theta_n}(\theta) = \frac{1}{2}\|\delta_n\|^2 + \mu S_{\theta_n}(\theta) \tag{10}$$

where $\mu > 0$ is a weighting constant and $S_{\theta_n}(\theta)$ is a function which quantifies change in the hidden units as a function of the change in parameters. It is given by

$$S_{\theta_n}(\theta) \equiv D(\, h(\theta)\, ;\, h(\theta_n)\,),$$

where D is a distance function similar to L. Note that we still include the usual Tikhonov term within this new damping function and so just as before, if λ becomes too high, all of the low curvature directions become essentially "masked out". However, with the inclusion of the $\mu S_{\theta_n}(\theta)$ term we hope that the quadratic model will tend to be more accurate at its optimum even for smaller values of λ and fortunately this is what we observed in practice, in addition to improved performance on the pathological long-term dependency problems.

When the damping function $R_{\theta_n}(\theta)$ is given by $\|\delta_n\|^2/2$, the function q_{θ_n} is a quadratic with respect to δ_n and thus CG can be directly applied with $\mathrm{B} = \mathrm{G}_f + \lambda I$ in order to minimize it. However, CG will not work with the structural damping function because S is not generally quadratic in θ. Thus, just as we approximated f by the quadratic function M, so too must we approximate S.

One option is to use the standard Taylor-series approximation:

$$S(\theta_n) + (\theta - \theta_n)^\top \left.\frac{\mathrm{d}S}{\mathrm{d}\theta}\right|_{\theta=\theta_n} + \frac{1}{2}(\theta - \theta_n)^\top \mathrm{H}_S(\theta - \theta_n)$$

where H_S is the Hessian of S at θ_n. Note that because D is a distance function, $D(h(\theta), h(\theta_n))$ is minimized at $\theta = \theta_n$ with value 0. This implies that

$$D(h(\theta), h(\theta_n)) = 0 \quad \text{and} \quad \frac{\mathrm{d}D(h(\theta), h(\theta_n))}{\mathrm{d}h(\theta)} = 0$$

Using these facts we can eliminate the constant and linear terms from the quadratic approximation of S since $S(\theta_n) = D(h(\theta_n), h(\theta_n)) = 0$ and

$$\begin{aligned}\left.\frac{\mathrm{d}S}{\mathrm{d}\theta}\right|_{\theta=\theta_n} &= \left.\frac{\mathrm{d}D(h(\theta), h(\theta_n))}{\mathrm{d}\theta}\right|_{\theta=\theta_n} \\ &= \left.\frac{\mathrm{d}D(h(\theta), h(\theta_n))}{\mathrm{d}h(\theta)}\frac{\mathrm{d}h(\theta)}{\mathrm{d}\theta}\right|_{\theta=\theta_n} = 0\end{aligned}$$

The fact that the gradient of the damping term is zero is important since otherwise including it with q_{θ_n} would result in a biased optimization process that could no longer be said to be optimizing the objective f.

Moreover, just as we did when creating a quadratic approximation of f, we may use the Gauss-Newton matrix G_S of S in place of the true Hessian. In particular, we define:

$$\mathrm{G}_S \equiv \mathrm{J}_{t,\theta}^\top (D \circ e)'' \mathrm{J}_{t,\theta}\big|_{\theta=\theta_n}$$

where $\mathrm{J}_{t,\theta}$ is the Jacobian of $t(\theta)$ w.r.t. θ (recall eq. 2: $h_i = e(t_i)$ and $(D \circ e)''$ is the Hessian of $D(e(t); h(\theta_n))$ w.r.t t). This matrix will be positive semi-definite if the distance function D matches the non-linearity e in the same way that we required L to match g in order to obtain the Gauss-Newton matrix for f.

And just as with G_f we may view the Gauss-Newton matrix for S as resulting from the Taylor-series approximation of the following convex approximation $\tilde{S}$ to S:

$$\tilde{S} = D(\, e(t(\theta_n) + \mathrm{J}_{t,\theta}\big|_{\theta=\theta_n}\delta_n)\, ;\, h(\theta_n)\,)$$

What remains is to find an efficient method for computing the contribution of the Gauss-Newton matrix for S to the usual curvature matrix-vector product to be used within the HF optimizer. While it almost certainly would require a significant amount of additional work to compute both $\mathrm{G}_f v$ and $\lambda\mu\mathrm{G}_S v$ separately, it turns out that we can compute their sum $\mathrm{G}_f v + \lambda\mu\mathrm{G}_S v = (\mathrm{G}_f + \lambda\mu\mathrm{G}_S)v$, which is what we actually need, for essentially the same cost as just computing $\mathrm{G}_f v$ by itself.

We can formally include $\lambda\mu S$ as part of the objective, treating $h = e(t)$ as an additional output of the network so that we linearize up to t in addition to s. This gives the combined Gauss-Newton matrix:

$$\mathrm{J}_{(s,t),\theta}^\top\Big|_{\theta=\theta_n} (L \circ g + D \circ e)''\ \mathrm{J}_{(s,t),\theta}\big|_{\theta=\theta_n}$$

where the inner Hessian is w.r.t. the concatenation of s and t (denoted (s,t)). Because the linearized prediction of t is already computed as an intermediate step when computing the linearized prediction of s, the required multiplication by the Jacobian $\mathrm{J}_{(s,t),\theta}|_{\theta=\theta_n}$ requires no extra work. Moreover, as long as we perform backpropagation starting from both s and t together (as we would when applying reverse-mode automatic differentiation) the required multiplication by the transpose Jacobian term doesn't either. The only extra computation required will be the multiplication by $(D \circ e)''$, but this is very cheap compared to the weight-matrix multiplication. Equivalently, we may just straightforwardly apply Pearlmutter's technique (involving a forward and backwards pass of the R-operator) for computing Hessian-vector products to any algorithm which efficiently computes $\tilde{f} + \lambda\mu\tilde{S}$. And said algorithm for computing $\tilde{f} + \lambda\mu\tilde{S}$ can itself be generated by applying a forward pass of R-operator to a function which computes s

and t (jointly, thus saving work), feeding the results into $L \circ g$ and $D \circ e$), respectively, and taking the finally taking the sum.

The result of applying either of these techniques is given as Algorithm 1 in the supplementary materials for specific choices of neural activation functions and error functions. Note that the required modification to the usual algorithm for computing the curvature matrix-vector product is simple and elegant and requires virtually no additional computation. However, it does require the extended storage of the Rt_i's, which could otherwise be discarded immediately after they are used to compute Rh_i.

While we have derived the method of 'structural damping' specifically for the hidden unit inputs t in the RNN model, it is easy to see that the idea and derivation generalize to any intermediate quantity z in any parameterized function we wish to learn. Moreover, the required modification to the associated matrix-vector product algorithm for computing the damped Gauss-Newton matrix would similarly be to add $\lambda\mu(D \circ e)'' Rz$ to z^* where D is the distance function that penalizes change in z.

There is valid concern that one can raise regarding the potential effectiveness of the structural damping approach. Because S is a highly non-linear function of δ_n it could be the case that the quadratic model will occasionally underestimate (or overestimate) the amount of change, perhaps severely. Moreover, if the linear model for the hidden units h is accurate for a particular δ_n then we probably don't need to penalize change in h, and if it's not accurate then structural damping may not help because, since it relies on the linear model, it could fail to detect any large changes in h. Fortunately, we can (mostly) address this concerns. First, even when the change in the hidden states (which are high-dimensional vectors) are not accurately predicted by the linear model, it could still be the case that the overall magnitude of the change is, since it's just a single scalar. Second, while the amount of change may be over or underestimated for any particular training case, we are averaging our computations over many (usually 1000s of training cases) and thus there may be some beneficial averaging effects. And third, the linear model detecting a large change in the hidden state sequence, while not a necessary condition for the break-down of the approximation, is at least a sufficient one (since we can't trust such large changes), and one which will be detected by structural damping term.

4. Experiments

In our first set of experiments we trained RNNs, using our HF-based approach, on various pathological synthetic problems known to be *effectively impossible* for gradient descent (e.g., Hochreiter et al., 2001). These problems were taken directly from the original LSTM paper (Hochreiter and Schmidhuber, 1997).

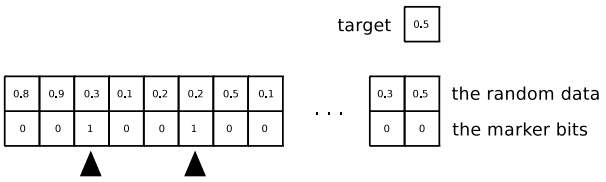

Figure 2. An illustration of the addition problem, a typical problem with pathological long term dependencies. The target is the sum of the two marked numbers (indicated with the black arrows). The ". . ." represent a large number of timesteps.

In our second set of experiments we trained both RNNs (using HF) and LSTMs[4] (using backprop-through-time) on more realistic high-dimensional time-series prediction tasks. In doing so we examine the hypothesis that the specialized LSTM approach, while being effective at tasks where the main difficulty is the need for long-term memorization and recall, may not be fully exploiting the power of recurrent computation in RNNs, and that our more general HF approach can do significantly better.

With a few exceptions we used the same meta-parameters and initializations over all of the experiments. See the paper supplement for details.

4.1. Pathological synthetic problems

In this section we describe the experiments we performed training RNNs with HF on seven different problems that all exhibit pathological long term dependencies. Each of these problems has a similar form, where the inputs are long sequences whose start (and possibly middle) are relevant, and the goal of the network is to output a particular function of the relevant inputs at the final few timesteps. The irrelevant part of the sequence is usually random, which makes the problem harder. A typical problem is illustrated in figure 2. The difficulty of these problems increases with T (the sequence length), since longer sequences exhibit longer range dependencies. Problems 1-6 are taken from Hochreiter and Schmidhuber (1997), so we will forgo giving a detailed description of them here.[5]

In our experiments, we trained RNNs with 100 hidden units using HF, with and without structural damping, on problems 1-7 for $T = 30, 50, 100, 200$. Our goal is to determine the amount of work the optimizer requires to train the RNN to solve each problem. To decide if the optimizer has succeeded we use the same criterion as Hochreiter and Schmidhuber.

1. **THE ADDITION PROBLEM**
In this problem, the input to the RNN is a sequence of random numbers, and its target output, which is located at the end of the sequence, is the sum of the two "marked" num-

[4]Specifically, the architecture that incorporates 'forget gates' which is described in Gers et al. (1999)

[5]For a more formal description of these problems and of the learning setup, refer to the supplementary materials.

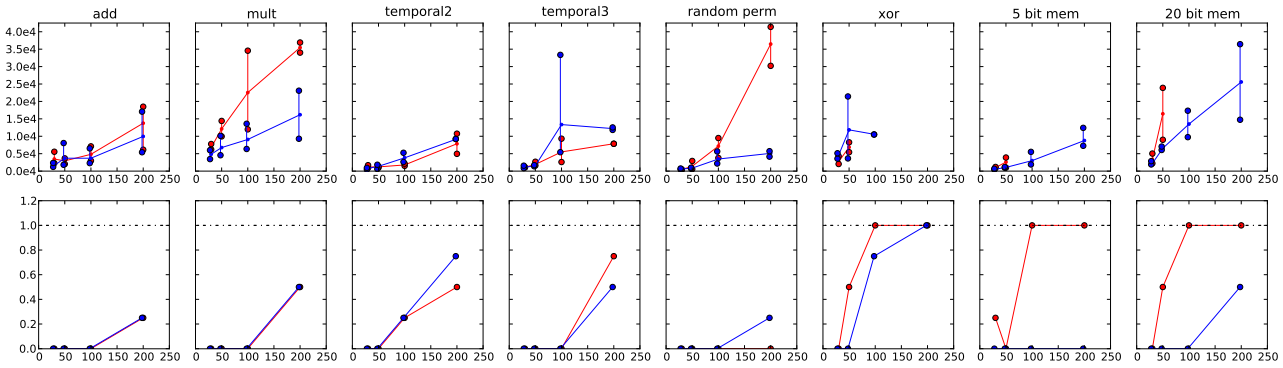

Figure 3. For each problem, the plots in the top row give the mean number of minibatches processed before the success condition was achieved (amongst trials that didn't fail) for various values of T (x-axis), with the min and max given by the vertical bars. Here, a minibatch is a set of 1000 sequences of length T that may be used for evaluating the gradient or a curvature matrix-vector product. A run using over 50,000 minibatches is defined as a failure. The failure rates are shown in the bottom row. Red is HF with only standard Tikhonov damping and blue is HF with structural damping included.

bers. The marked numbers are far from the sequence's end, their position varies, and they are distant from each other. We refer to Hochreiter and Schmidhuber (1997) for the formal description of the addition problem, and to fig. 2 for an intuitive illustration.

2. **The multiplication problem**
The multiplication problem is precisely analogous to the addition problem except for the different operation.

3. **The Xor problem**
This problem is similar to the addition problem, but the noisy inputs are binary and the operation is the Xor. This task is difficult for both our method and for LSTMs (Hochreiter and Schmidhuber, 1997, sec. 5.6) because the RNN cannot obtain a partial solution by discovering a relationship between one of the marked inputs and the target.

4. **The temporal order problem**
See Hochreiter and Schmidhuber (1997, task 6a).

5. **The 3-bit temporal order problem**
See Hochreiter and Schmidhuber (1997, task 6b).

6. **The random permutation problem**
See Hochreiter and Schmidhuber (1997, task 2b).

7. **Noiseless memorization**
The goal of this problem is to learn to memorize and reproduce long sequences of bits. The input sequence starts with a string of 5 bits followed by T occurrences of a constant input. There is a target in every timestep, which is constant, except in the last 5 timesteps, which are the original 5 bits. There is also a special input in the 5th timestep before last signaling that the output needs to be presented. We also experimented with a harder variant of this problem, where the goal is to memorize and sequentially reproduce 10 random integers from $\{1, \ldots, 5\}$ which contain over 20 bits of information.

Results and Discussion

The results from our experiments training RNNs with HF on the pathological synthetic problems are shown in figure 3. We refer the reader to Hochreiter and Schmidhuber (1997) for a detailed analysis of the performance of LSTMs on these tasks. Our results establish that our HF approach is capable of training RNNs to solve problems with very long-term dependencies that are known to confound approaches such as gradient descent. Moreover, the inclusion of structural damping adds some additional speed and robustness (especially in the 5 and 20 bit memorization tasks, where it appears to be required when the minimal time-lag is larger than 50 steps). For some random seeds the exploding/vanishing gradient problem seemed to be too extreme for the HF optimizer to handle and the parameters would converge to a bad local minimum (these are reported as failures in figure 3). This is a problem that could potentially be addressed with a more careful random initialization than the one we used, as evidenced by the significant amount of problem-specific hand tweaking done by Hochreiter and Schmidhuber (1997, Table 10).

It should be noted that the total amount of computation required by our HF approach in order to successfully train an RNN is considerably higher than what is required to train the LSTM using its specialized variant of stochastic gradient descent. However, the LSTM architecture was designed specifically so that long-term memorization and recall can occur naturally via the "memory units", so representationally it is a model ideally suited to such tasks. In contrast, the problem of training a standard RNN to succeed at these tasks using only its general-purpose units, which lack the convenient "gating neurons" (a type of 3-way neural connection), is a *much* harder one, so the extra computation required by the HF approach seems justified. And it should also be noted that the algorithm used to train LSTMs is fully online and thus doesn't benefit from parallel computation nearly as much as a semi-online approach such as

Table 1. Average test-set performance for HF-trained RNN and the LSTM on the three natural sequence modeling problems.

Dataset (measure)	RNN+HF	LSTM+GD
Bouncing Balls (error)	22	35
MIDI (log-likelihood)	-569	-683
Speech (error)	22	41

HF (where the training cases in a given mini-batch may be processed all in parallel).

4.2. Natural problems

For these problems, the output sequence is equal to the input sequence shifted forward by one time-step, so the goal of training is to have the model predict the next timestep by either minimizing the squared error, a KL-divergence, or by maximizing the log likelihood. We trained an RNN with HF and an LSTM with backpropagation-through-time (verified for correctness by numerical differentiation). The RNN had 300 hidden units, and the LSTM had 30 "memory blocks" (each of which has 10 "memory cells") resulting in nearly identically sized parameterizations for both models (the LSTM had slightly more). Full details of the datasets and of the training are given in the supplementary material.

1. **BOUNCING BALL MOTION VIDEO PREDICTION**
The bouncing balls problem consists of synthetic low-resolution (15×15) videos of three balls bouncing in a box. We trained the models on sequences of length 30.

2. **MUSIC PREDICTION**
We downloaded a large number of midi files and converted each into a "piano-roll". A piano-roll is a sequence of 128-dimensional binary vectors, each describing the set of notes played in the current timestep (which is 0.05 seconds long). The models were trained on sequences of length 200 due to the significant long-term dependencies exhibited by music.

3. **SPEECH PREDICTION**
We used a speech recognition dataset to obtain sequences of 39-dimensional vectors representing the underlying speech signal, and used sequences of length 100.

RESULTS
The results are shown in table 1. The HF-trained RNNs outperform the LSTMs trained with gradient descent by a large margin. It should be noted that training the LSTMs took just as long or longer than the RNNs.

5. Conclusions

In this paper, we demonstrated that the HF optimizer of Martens (2010), augmented with our structural damping approach, is capable of robustly training RNNs to solve tasks that exhibit long term dependencies. Unlike previous approaches like LSTMs, the method is not specifically designed to address the underlying source of difficulty present in these problems (but succeeds nonetheless), which makes it more likely to improve RNN training for a wider variety of sequence modeling tasks that are difficult, not only for their long term dependencies, but also for their highly complex shorter-term evolution, as our experiments with the natural problems suggest. Given these successes we feel that RNNs, which are very powerful and general sequence models in principle, but have always been very hard to train, may now begin to realize their true potential and see more wide-spread application to challenging sequence modeling problems.

Acknowledgments
The authors would like to thank Richard Zemel and Geoffrey Hinton for their helpful suggestions. This work was supported by NSERC and a Google Fellowship.

REFERENCES

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

F.A Gers, J. Schmidhuber, and Cumminsm F. Learning to forget: Continual prediction with lstm. *Neural Computation*, 12:2451–2471, 1999.

A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18:602–610, 2005.

A. Graves and J. Schmidhuber. Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks. In *Advances in Neural Information Processing Systems*, 2009.

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

S. Hochreiter. *Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis*. PhD thesis, Institut fur Informatik, Technische Universitat Munchen, 1991.

S. Hochreiter and J. Schmidhuber. Bridging long time lags by weight guessing and "long short term memory". In *Spatiotemporal models in biological and artificial systems*, 1996.

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

S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. *A Field Guide to Dynamical Recurrent Neural Networks*, chapter Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. IEEE press, 2001.

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

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

H. Mayer, F. Gomez, D. Wierstra, I. Nagy, A. Knoll, and J. Schmidhuber. A system for robotic heart surgery that learns to tie knots using recurrent neural networks. In *2006 International Conference on Intelligent Robots and Systems*, 2007.

K.P. Murphy. *Dynamic bayesian networks: representation, inference and learning*. PhD thesis, UC Berkeley, 2002.

J. Nocedal and S.J. Wright. *Numerical optimization*. Springer Verlag, 1999.

B.A. Pearlmutter. Fast exact multiplication by the Hessian. *Neural Computation*, 6(1):147–160, 1994.

D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088): 533–536, 1986.

N. Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 14(7): 1723–1738, 2002.