Learning Recurrent Neural Networks with Hessian-Free Optimization: Supplementary Materials

Contents

1 Pseudo-code for the damped Gauss-Newton vector product

Algorithm 1 Computation of the matrix-vector product of the structurally-damped Gauss-Newton matrix with the vector v, for the case when e is the $\tanh$ non-linearity, g the logistic sigmoid, D and L are the corresponding matching loss functions. The notation reflects the "convex approximation" interpretation of the GN matrix so that we are applying the R operator to the forwards-backwards pass through the linearized and structurally damped objective $\tilde{k}$, and the desired matrix-vector product is given by $\mathrm{R}\frac{\mathrm{d}\tilde{k}}{\mathrm{d}\theta}$. All derivatives are implicitly evaluated at $\theta = \theta_n$. The previously defined parameter symbols W_{ph}, W_{hx}, W_{hh}, b_h, b_p b_h^{init} will correspond to the parameter vector θ_n if they have no super-script and to the input parameter vector v if they have the 'v' superscript. The $\mathrm{R}z$ notation follows Pearlmutter [1994], and for the purposes of reading the pseudo-code can be interpreted as merely defining a new symbol. We assume that intermediate quantities of the network (e.g. h_i) have already been computed (from θ_n). The operator $\odot$ is coordinate-wise multiplication. Line 17 (underlined) is responsible for structural damping.

```
1:  for $i = 1$ to $T$ do
2:      if $i = 1$ then
3:          $\mathrm{R}t_i \leftarrow b_h^v + b_h^{\text{init}\,v} + W_{hx}^v x_i$
4:      else
5:          $\mathrm{R}t_i \leftarrow b_h^v + W_{hx}^v x_i + W_{hh}^v h_{i-1} + W_{hh}\mathrm{R}h_{i-1}$
6:      end if
7:      $\mathrm{R}h_i \leftarrow (1 + h_i) \odot (1 - h_i) \odot \mathrm{R}t_i$
8:      $\mathrm{R}s_i \leftarrow b_p^v + W_{ph}^v h_i + W_{ph}\mathrm{R}h_i$
9:      $\mathrm{R}\hat{y}_i \leftarrow \hat{y}_i \odot (1 - \hat{y}_i) \odot \mathrm{R}s_i$
10: end for
11: $\mathrm{R}\frac{\mathrm{d}\tilde{k}}{\mathrm{d}\theta} \leftarrow 0$
12: $\mathrm{R}\frac{\mathrm{d}\tilde{k}}{\mathrm{d}t_{T+1}} \leftarrow 0$
13: for $i = T$ down to 1 do
14:     $\mathrm{R}\frac{\mathrm{d}\tilde{k}}{\mathrm{d}s_i} \leftarrow \mathrm{R}\hat{y}_i$
15:     $\mathrm{R}\frac{\mathrm{d}\tilde{k}}{\mathrm{d}h_i} \leftarrow W_{hh}^\top \mathrm{R}\frac{\mathrm{d}\tilde{k}}{\mathrm{d}t_{i+1}} + W_{ph}^\top \mathrm{R}\frac{\mathrm{d}\tilde{k}}{\mathrm{d}s_i}$
16:     $\mathrm{R}\frac{\mathrm{d}\tilde{k}}{\mathrm{d}t_i} \leftarrow (1 + h_i) \odot (1 - h_i) \odot \mathrm{R}\frac{\mathrm{d}\tilde{k}}{\mathrm{d}h_i}$
17:     $\underline{\mathrm{R}\frac{\mathrm{d}\tilde{k}}{\mathrm{d}t_i} \leftarrow \mathrm{R}\frac{\mathrm{d}\tilde{k}}{\mathrm{d}t_i} + \lambda\mu(1 + h_i) \odot (1 - h_i) \odot \mathrm{R}t_i}$
18:     $\mathrm{R}\frac{\mathrm{d}\tilde{k}}{\mathrm{d}W_{ph}} \leftarrow \mathrm{R}\frac{\mathrm{d}\tilde{k}}{\mathrm{d}W_{ph}} + \mathrm{R}\hat{y}_i h_i^\top$
19:     $\mathrm{R}\frac{\mathrm{d}\tilde{k}}{\mathrm{d}W_{hh}} \leftarrow \mathrm{R}\frac{\mathrm{d}\tilde{k}}{\mathrm{d}W_{hh}} + \mathrm{R}\frac{\mathrm{d}\tilde{k}}{\mathrm{d}t_{i+1}} h_i^\top$
20:     $\mathrm{R}\frac{\mathrm{d}\tilde{k}}{\mathrm{d}W_{hx}} \leftarrow \mathrm{R}\frac{\mathrm{d}\tilde{k}}{\mathrm{d}W_{hx}} + \mathrm{R}\frac{\mathrm{d}\tilde{k}}{\mathrm{d}t_i} x_i^\top$
21:     $\mathrm{R}\frac{\mathrm{d}\tilde{k}}{\mathrm{d}b_h} \leftarrow \mathrm{R}\frac{\mathrm{d}\tilde{k}}{\mathrm{d}b_h} + \mathrm{R}\frac{\mathrm{d}\tilde{k}}{\mathrm{d}t_i}$
22:     $\mathrm{R}\frac{\mathrm{d}\tilde{k}}{\mathrm{d}b_p} \leftarrow \mathrm{R}\frac{\mathrm{d}\tilde{k}}{\mathrm{d}b_p} + \mathrm{R}\hat{y}_i$
23: end for
24: $\mathrm{R}\frac{\mathrm{d}\tilde{k}}{\mathrm{d}b_h^{\text{init}}} \leftarrow \mathrm{R}\frac{\mathrm{d}\tilde{k}}{\mathrm{d}t_1}$
```

2 Details of the pathological synthetic problems

We begin by describing the experimental setup that was used in all the pathological problems. In every experiment, the RNN had 100 hidden units and a little over 10,000 parameters. It was initialized with a sparse initialization [Martens, 2010]: each weight matrix (W_{hx}, W_{hh}, and W_{ho}) is sparsely initialized so that each unit is connected to 15 other units. The nonzero connections of W_{hh}, of W_{ho}, and the biases are sampled independently from a Gaussian with mean 0 and variance $\frac{1}{15}$, while the nonzero connections of W_{hx} are independently sampled from a Gaussian with mean 0 and a variance of 1. These constants were chosen so that the initial gradient wouldn't vanish or explode too strongly for the optimizer to cope with. The occasional failures for some random seeds of the HF approach on a few of the harder synthetic problems was likely due to a deficiency in this initialization scheme and we are currently investigating ones that may be more robust.

The gradient was computed on 10,000 sequences, of which 1,000 were used to evaluate the curvature matrix-vector products (except for the 5-bit version of problem 7, where there are only 32 possible distinct sequences, so the gradient and the curvature matrix-vector products were computed using all the data). Every HF iteration used a fresh set of sequences and the test set also consists of 10,000 sequences. We allowed for at most 300 steps of CG at each run of HF, but we would stop the CG run sooner if the magnitude of the reduction in the objective function on the curvature minibatch is less than 90% of its maximum during the CG run. The damping coefficient λ was initialized to 0.1 if structural damping was used in which case μ was set to $1/30$, and to 0.3 otherwise.

The criteria for success and failure we used were taken directly from Hochreiter and Schmidhuber [1997]. In particular, a run is defined to be successful if less than 1% of the test sequences are misclassified. While the notion of sequence misclassification is trivially defined for binary or discrete problems, for continuous problems (such as the addition or the multiplication problem), we say that a sequence is misclassified if the absolute prediction error exceeds 0.04.

In some of the problems, the inputs or the outputs are symbols. Input symbols are simply represented with their 1-of-N encoding. To predict a symbol, the RNN uses the softmax output nonlinearity with the cross entropy loss function, whose composition is "matching" and therefore the Gauss-Newton matrix can be used.

In the following subsections, we let $U[a, b]$ denote a uniformly random real number from the interval $[a, b]$ and $i[a, b]$ denote a uniformly random integer from the same interval. The descriptions also assume that a desired problem length T has been selected.

2.1 The addition, multiplication, and XOR problem

The addition problem is described in Figure 2 of the paper. The inputs consist of sequences of random numbers, and the target output, which is located in the end of the sequence, is the sum of the two "marked" numbers. The marked numbers are far from the sequence's end, their position varies, and they are distant from each other.

To generate a single training sequence for the addition problem, let the length of the sequence T' be a sample from $i[T, 11T/10]$ (so that $T' > T$). Let I and J be the positions of the marked inputs, which are drawn from $i[1, T'/10]$ and let $i[T'/10, T'/2]$. The input x will consist of the pairs (u, v) where the u_t's are independently drawn from $U[0, 1]$ and the "markers" v_t will be 0, except at $t = I$ and J, where $u_I = u_J = 1$. Finally, let the target output at time T' be the normalized sum $(v_I + v_J)/2$. Thus the u_t component of the input $x_t = (u_t, v_t)$ is irrelevant when t is not I or J.

The multiplication and the XOR problem are completely analogous.

Note that these problems are challenging for the RNN not only because of their significant long term dependencies, but also because the RNN's hidden state is exposed to both the relevant and the irrelevant inputs at each timestep and must learn to accurately remember the former whilst ignoring the latter. This is a difficult problem for the RNN, which, unlike the LSTM, does not have a built-in means to "protect" its hidden state from irrelevant inputs in the form of the "gating units". Yet as we saw, our methods can solve this and related tasks for T as large as 200.

2.2 The temporal order problem

In this task [task 6a in Hochreiter and Schmidhuber, 1997], the input sequences consist of random symbols in $\{1, 2, ..., 6\}$ which are represented with 1-of-6 encodings. All of the inputs are irrelevant and are the encodings of randomly chosen integers in $\{3, 4, 5, 6\}$ except for 2 special ones which are encodings of randomly chosen integers in $\{1, 2\}$. The target output is a 1-of-4 encoding of the ordered-pair of the two special symbols.

Formally, let I and J be the positions of the special inputs which are drawn from $i[T/10, 2T/10]$ and $i[5T/10, 6T/10]$ respectively. We let $z_1, \dots, z_T$ be independent draws from $i[3, 6]$, and let a, b, the special symbols, be independently drawn from $i[1, 2]$. Let e_j denote the 1-of-6 encoding of j. Then we set the input x_t to e_{z_t} for $t \neq I$ or J, and set x_I to e_a and x_J to e_b. The target output at time T is the pair (a, b) which is represented with a 1-of-4 encoding (because there are four possibilities for (a, b)).

2.3 The 3-bit temporal order problem

This problem is similar to the above, except that there are 3 special symbols in the input sequence, and thus 8 possible outputs [see Hochreiter and Schmidhuber, 1997, task 6b, for more details].

The formal setup is nearly identical to the previous one. The positions I, J, and K of the three special symbols are drawn from $i[T/10, 2T/10]$, $i[3T/10, 4T/10]$, and $i[6T/10, 7T/10]$, respectively. The values of the special symbols a, b, c are independently drawn from $i[1, 2]$. Finally, $x_I = e_a, x_J = e_b, x_K = e_c$, and the target is (a, b, c) represented with a 1-of-8 encoding.

2.4 The random permutation problem

The inputs consist of sequences of symbols from $\{1, \dots, 100\}$. The first and the last symbol are identical and their value is drawn from $i[1, 2]$, while the rest of the symbols are independently drawn from $i[3, 100]$. At each timestep, the target output is equal to the input symbol of the next timestep, and as a result only the first and the last symbol are predictable. This problem is difficult not only for its long term dependencies, but also because the loss incurred at the last timestep (the one we use to measure success) is small compared to the loss associated with the other timesteps, which may cause the optimizer to focus on an incorrect aspect of the problem.

In this task, we initialized λ to $0.1 \cdot T$ because there are many more outputs which make the scale of the objective is considerably larger.

2.5 Noiseless memorization

In this problem, the input sequence starts with a string of 5 bits followed by T occurrences of a constant input. The target output is constant up until the last 5 timesteps, which are the original 5 input bits. There is also a special input in the 5th-last timestep signaling that the output needs to be presented.

In the following, inputs and the outputs are represented by 1-of-K vector encodings but for brevity we describe them as symbols. More formally, let $x_1, \ldots, x_{T+10}$ be the input symbols and $y_1, \ldots, y_{T+10}$ be the target symbols. The sequence begins with $x_1, \ldots, x_5$ which are drawn from $i[1, 2]$ and are thus binary. The subsequent inputs are constant except at $t = T + 5$, which is a "trigger" symbol (set to 4) which signals that the memorized symbols should be produced as output. The target outputs are obtained by setting $y_j = 3$ for $j \leq T + 5$, and $y_{j+T+5} = x_j$ for $1 \leq j \leq 5$. In particular, most of the predictions (up to timestep $T + 5$) are irrelevant.

We straightforwardly adapt the problem of memorizing 10 symbols from $\{1, \ldots, 5\}$.

3 Details of the natural problems

In all 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 now describe the details of the natural datasets and the settings of the optimizer used.

3.1 The bouncing balls problem

This dataset consists of synthetically generated videos of three balls bouncing in a box. Given that the sequences can be perfectly predicted using only a small number of previous timesteps, we trained both models on sequences of length 30. The resolution of the videos is 18×18.

The pixels are real values between 0 and 1, so the RNN and the LSTM are trained to minimize the per-pixel KL divergence $\sum_i q_i \log q_i/p_i + (1 - q_i) \log{(1 - q_i)}/(1 - p_i)$, where q_i is the ith pixel and p_i is the ith prediction (which is the output of the sigmoid function). This also is the per-sequence error measure we reported in Table 1 in the paper.

For training of the RNN with the HF approach we used $4 \cdot 10^4$ sequences for computing the gradient and $4 \cdot 10^3$ for computing the curvature matrix-vector products. We

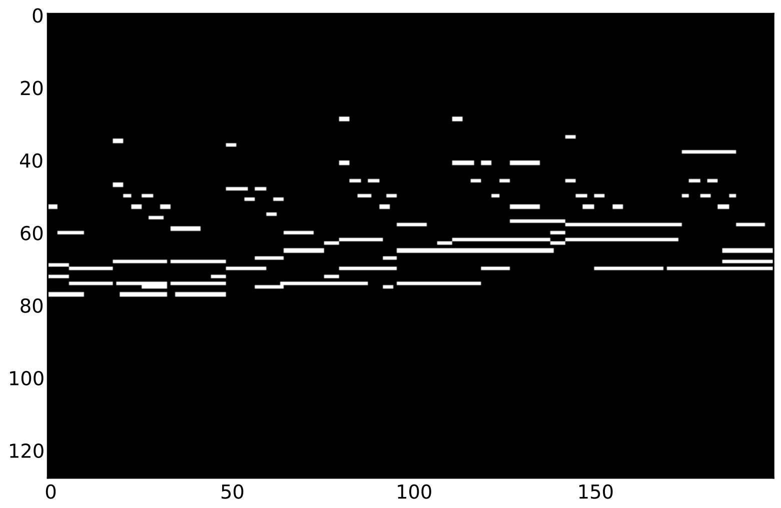

Figure 1: An example of a training sequence from the MIDI music dataset. The sequence is 200 timestep long, and each timestep is a 128-dimensional binary vector. Typical sequences from this dataset exhibit a substantial amount of nontrivial long range temporal dependencies.

stopped training after a total of $100 \cdot 10^6$ sequences (not necessarily distinct) had been processed. In this experiment, μ was set to 1.0 and λ was initialized to 1.

The LSTM was trained with stochastic gradient descent with a learning rate of 0.001 and a momentum of 0.9 for a total of 10^5 weight updates, where each gradient was computed as the average over a minibatch of 50 training sequences. It was initialized with an analogous sparse initialization, where the nonzero entries were sampled independently from a Gaussian with mean 0 and variance 0.01. We also added 3.0 to the biases of the "forget gates" which we found was beneficial to the optimization.

3.2 The MIDI dataset

To construct this dataset we downloaded 3364 MIDI files from an online database and created a training set consisting of 100,000 sequences of length 200, where each timestep corresponds to the set of tones present in a 0.05-second long interval (represented as a 128-dimensional binary vector). Thus each sequence corresponds to exactly 10 seconds of music. We increased the size of the dataset by translating the tones in each sequence by an offset sampled from $i[-10, 10]$. Figure 1 depicts a typical sequence from this dataset.

Due to the binary nature of the data, we trained the RNN and LSTM to maximize the log probability of the sequences using the cross-entropy objective $\sum_i t_i \log p_i + (1 - t_i) \log(1 - p_i)$, where t_i ranges over the dimensions of the timestep and p_i is the set of predictions. For training of the RNN with the HF approach we used $2 \cdot 10^4$

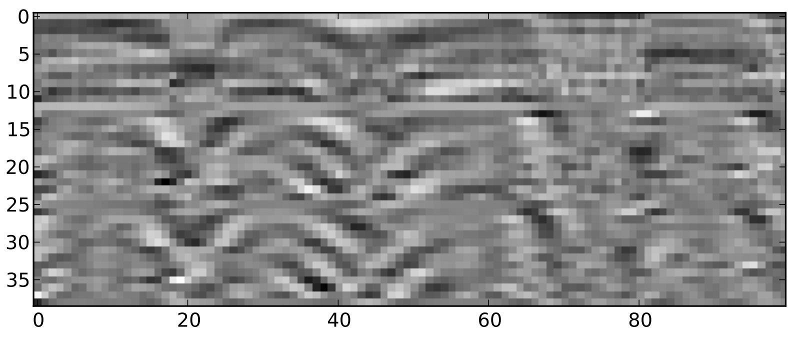

Figure 2: An example of a training sequence from the speech dataset. The sequence is 100 timestep long, and each timestep is a 39-dimensional real valued vector.

sequences for computing the gradient and $2 \cdot 10^3$ for computing the curvature matrix-vector products. We stopped training after a total of $60 \cdot 10^6$ sequences had been processed. We initialized λ to 100 and set μ to 0.1. The LSTM was trained for $5 \cdot 10^4$ gradient steps with a learning rate of 0.001, where each gradient minibatch was computed on 50 sequences, and was otherwise equivalent to the previous LSTM.

3.3 The speech dataset

We used the TIMIT dataset [Garofolo et al., 1993] to construct the speech timeseries. The training sequences are 100 timesteps long, where each timestep is a 39-dimensional vector of the Mel-frequency cepstral coefficients (MFCC) representation of the signal (see fig. 2 for an example sequence). As the data is real-valued, we trained the RNN and LSTM to minimize the squared error.

For training of the RNN with the HF approach we used $2 \cdot 10^4$ sequences for computing the gradient and $2 \cdot 10^3$ for computing the curvature matrix-vector products. We stopped training after a total of $3 \cdot 10^6$ had been processed. We initialized λ to 50 and set μ to 0.1. The LSTM was trained for 10^5 gradient steps with a learning rate of 0.001, where each gradient minibatch was computed on 50 sequences, and was initialized as in the other experiments.

References

J.S. Garofolo, L.F. Lamel, W.M. Fisher, J.G. Fiscus, D.S. Pallett, and N.L. Dahlgren. *Darpa Timit: Acoustic-phonetic Continuous Speech Corps CD-ROM*. US Dept. of Commerce, National Institute of Standards and Technology, 1993.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

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

B.A. Pearlmutter. Fast exact multiplication by the Hessian. *Neural Computation*, 1994.