Diagnet: A New Recurrent Neural Network Architecture

Thomas Lahore
tom.lahore@gmail.com

Morgan Weaver
morganjweaver@gmail.com

Abstract

Recurrent Neural Networks have long been plagued by the vanishing and exploding gradient problems and limited performance over increasing time steps. We propose Diagnet, a novel RNN design which incorporates architectural and process-oriented elements which, in combination, greatly increase the stability and quality of the error gradient. We demonstrate in a variety of benchmarking tasks that learning becomes possible with delays of over 10,000 time steps, representing substantial gains over existing RNN designs. Diagnet's architecture relies upon three core features: 1) A single layer of segregated recurrent units which are self-connected 2) The absolute value function (an unbounded norm-preserving idempotent non-linearity), and 3) Application of constraints in both recurrent parameter weights and gradient norms, which tightly control the exploding gradient problem. Despite the segregated and simplistic nature of Diagnet's recurrent units, significant performance improvement is seen in the 1000-example version of the bAbI task over LSTM and GRU. In addition, Diagnet scores higher in testing accuracy on 15 out of bAbI 20 bAbI tasks, by margins of 3-30%, while supporting more hidden layers than many other RNN architectures.

1 Introduction

Importance of RNNs Recurrent Neural Networks represent one of the most important deep learning architectures in current use. They are used for many tasks, such as time series prediction [REF] , unsegmented, connected handwriting recognition [REF], large-vocabulary speech recognition [REF], modeling chaotic phenomena [REF], language modeling and machine translation [REF], robot control [REF], Protein Homology Detection [REF], difficulties of training RNNs [REF], and vanishing/exploding gradient problems [REF]. Wikipedia: "A major problem with gradient descent for standard RNN architectures is that error gradients vanish exponentially quickly with the size of the time lag between important events.[34][67] LSTM combined with a BPTT/RTRL hybrid learning method attempts to overcome these problems.[6] " Vanishing gradients due to the nonlinearities used (e.g. ReLU) Problem space We are interested in addressing two widely-cited [RED] pain points of RNN performance. The first being the vanishing and exploding gradient problem [REF], and the second being performance over increasing time steps [REF]. We address these issues by implementing three as kjfhakdf that have not yet been exibited in combination in neural net design. What we found Results were...impressive Paper Outline in brief In the following paper, we present a brief outline of foundational work in the problem space to contextualize Diagnet's emergence in RNN design. Then, we present a discussion of the architecture and mechanisms of Diagnet, along with a useful visual tool for building an intuitive understanding of Diagnet's units and their behavior. Next we present a series of common benchmarking tasks, the results of each task, and finally conclude with a discussion of Diagnet's performance and behavior based on experimental results. We hope to offer an interesting and powerful new tool for sequence modeling in our introduction of Diagnet, while laying the groundwork for further exploration, architectural innovation, and interaction with the ML community.

Preprint. Work in progress.

1.1 Style

Papers to be submitted to NIPS 2018 must be prepared according to the instructions presented here. Papers may only be up to eight pages long, including figures. Additional pages *containing only acknowledgments and/or cited references* are allowed. Papers that exceed eight pages of content (ignoring references) will not be reviewed, or in any other way considered for presentation at the conference.

The margins in 2018 are the same as since 2007, which allow for $\sim 15\%$ more words in the paper compared to earlier years.

Authors are required to use the NIPS LaTeX style files obtainable at the NIPS website as indicated below. Please make sure you use the current files and not previous versions. Tweaking the style files may be grounds for rejection.

1.2 Retrieval of style files

The style files for NIPS and other conference information are available on the World Wide Web at

`http://www.nips.cc/`

The file `nips_2018.pdf` contains these instructions and illustrates the various formatting requirements your NIPS paper must satisfy.

The only supported style file for NIPS 2018 is `nips_2018.sty`, rewritten for LaTeX 2_ε. **Previous style files for LaTeX 2.09, Microsoft Word, and RTF are no longer supported!**

The LaTeX style file contains three optional arguments: `final`, which creates a camera-ready copy, `preprint`, which creates a preprint for submission to, e.g., arXiv, and `nonatbib`, which will not load the `natbib` package for you in case of package clash.

New preprint option for 2018 If you wish to post a preprint of your work online, e.g., on arXiv, using the NIPS style, please use the `preprint` option. This will create a nonanonymized version of your work with the text "Preprint. Work in progress." in the footer. This version may be distributed as you see fit. Please **do not** use the `final` option, which should **only** be used for papers accepted to NIPS.

At submission time, please omit the `final` and `preprint` options. This will anonymize your submission and add line numbers to aid review. Please do *not* refer to these line numbers in your paper as they will be removed during generation of camera-ready copies.

The file `nips_2018.tex` may be used as a "shell" for writing your paper. All you have to do is replace the author, title, abstract, and text of the paper with your own.

The formatting instructions contained in these style files are summarized in Sections 2, 3, and 4 below.

2 Previous Work

It is important to consider a new deep learning architecture with an awareness of existing models. We offer a brief and by no means comprehensive overview of existing RNN designs herein. Some of the earliest work consists of Hopfield Networks, developed by Hopfield in 1982. Subsequently, Schmidhuber [REF] developed the Neural History Compressor (1993) and was able to preserve information over time steps, while compressing information. In 1991 Fallman developed the

Recurrent Cascade Correlation Algorithm (1991)–Fallman Add 1 hidden neuron at a time. All else frozen. Inputs go here. Output of prev goes to new and also all inputs. Each new hidden neuron must be maximally correlated with current remaining err per step. Now make recurrent: each neuron has single link to self. Rinse n repeat algo; truncated backprop at the time bc feasible. At time, proved not useful for certain grammars like seq parity (using saturating, not abs val fxn) First paper T knows of where neuron self-connected. (IndRNN first layer is the same) LSTM (1997)-gradient preservation. first really good thing for v/exp gradient problem. Forgets/gates. GRU (2014)-just ref it, no significant influence IRNN (2015)-identity RNN–HINTON, recurrent neurons are relus, init at

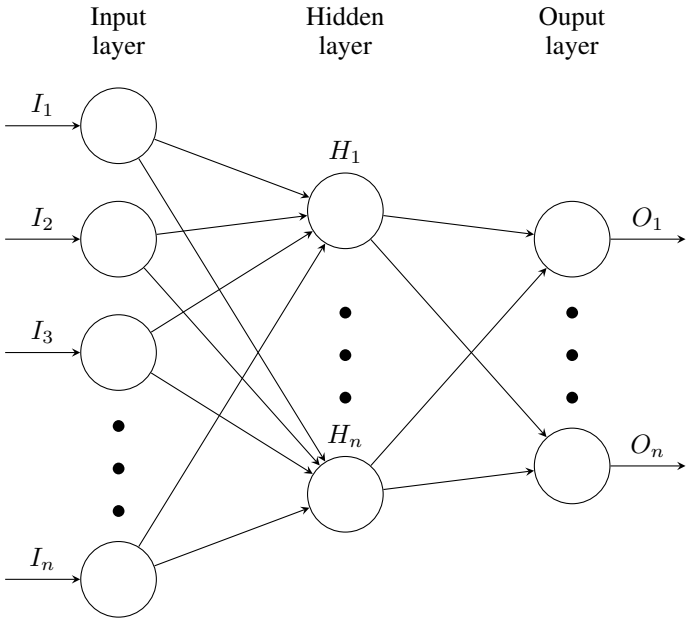

Figure 1: Figure showing nice diagram of the architecture.

ID matrix. 2 big ideas. Prob: not used bc unstable, need right params. Good init results > LSTM simpler architecture. Init at ID matrix was the big idea (unitary matrix) Diagnet too–but off-dig values can't ever be anything but 0; diag are all 1 fast Hadamard product instead of tiresome n$\hat{2}$. SCRN (2015)–diagonal entries connected constrained to 1 or a min val, so self-recurrent. Recognizing value of separate way of handling given recurrent unit connections to self. Better task results but not less complex computationally. Higher quality learning process. uRNN (2015)-Bengio et al. addresses vanishing gradient prob, not exploding. Transition matrix is unitary. Enorced. Complex numbers, etc. Decent results. DizzyRNN (2016)-abs val. weight matrix unitary transformations, broken into pairs of 2, use 2d rotations. reasonable results. 1-2 tasks IndRNN (2018)–uses self-connection in first layer, loses much efficiency. Many-to-many connection. Very expensive. Uses relu for non-linearity rather than our absval. Gradient clipping–2 uses here–1)take norm of gradient over all params and limit to N; throw away. If larger, rescale. 2) Gradients on indiv params, scale down if exceeds limit. Lossy, loses vector direction information. Directionality is better preserved in this multi-targeted approach. First pass then prune outliers.

Besides architectural innovations, progress has been made in controlling the gradient directly.

Clipped Gradients (Paskanu, Mikolov, Bengio)

Make a chart of architectures vs features? Make a feature matrix showing what overlaps the various solutions have? That might be of more interest, and faster than simply talking about all of them individually. We can still cite them.

3 Architecture

Densely connected from input to hidden layer. Only a single, "shallow" hidden layer. Hidden layer has no biases. Hidden layer is recurrently connected by a diagonal matrix; thus every hidden neuron is solely recurrently self-connected. Self connections of hidden neurons are clamped to lie in the range [-1.0, 1.0] Self connections of hidden neurons are initialized at 1.0. Each hidden neuron makes use of the absolute value function as a nonlinearity. There are no gates* (see "poor man's gate" section later for an architectural variant) Densely connected from hidden layer to output

3.1 Absolute Value Function

The absolute value function was chosen because it is has a number of desirable properties: It is nonlinear (a requirement for interesting computations to be possible) It is unbounded. (Can I make an argument for why this is an important thing? Something somthing Turing-complete?) It is idempotent! (This aspect of stability may allow for long term, high precision storage of information without specialized architectural design) It is (almost) everywhere differentiable, with unit-length derivatives of either -1 or 1. It is norm-preserving (and hence gradient preserving), both for each individual component of the vector, and also for the entire vector. The absolute value function cannot, by itself, cause gradients to explode, nor vanish.

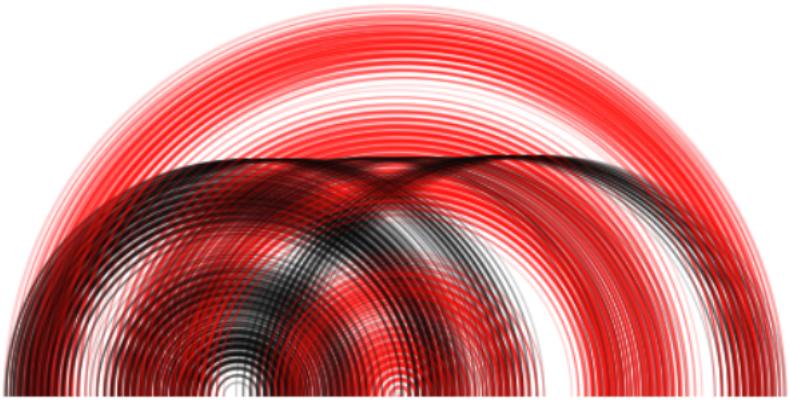

Figure 2: Absolute value dynamics

Generic restatements about why norm preservation is important when training recurrent neural networks.

LSTM and GRU preserve the gradient over time by mostly avoiding the direct application of nonlinearities as much as possible. (ResNet is similar in this sense.) However, we speculate that many potentially useful algorithms or behaviors likely require the application of a very large number of computations/transformations, rather than simply preserving a small number of computations/transformations over large spans of time.

One toy problem that exhibits such a computational requirement is sequential parity, which will be visited below (and which LSTM and GRU are completely unable to solve for T > 10.)

Others (although surprisingly few) have experimented with using the absolute value function in RNNs, see [REF], [REF]... We speculate that it has not received much attention as of yet, because very special care must be taken to control the spectral radius of the recurrent matrix.

Something about absolute value function being the only continuous norm preserving non-linearity (is that true?). Something about nice properties of absolute value function. Idempotent, etc..

3.2 Constraints

Postulate that absolute value function hasn't been explored much, particularly for recurrent neural nets, because without the proper constraints, its values tend to rapidly blow up.

3.2.1 Limited [-1, 1] range

Important to have these constraints, otherwise blows up. This is an extremely simple way of limiting the spectral radius of the matrix. In fact (double check) because of the constrained structure, these "recurrent factors" can straightforwardly tell us the singular values, eigenvalues, spectral radius of the linear transformation. (I think the fact that we use no biases helps because it remains linear rather than affine.)

$$h_t := |u \circ h_{t-1} + W\sigma(Vx_t))|$$

Figure 3: Hidden state update formula.

Mention that on some tasks, it often ends up using very different "signatures" of recurrent factor values.

Mention importance of initializing to 1.0 here? Different section?

3.2.2 Two complementary types of gradient clipping

Some form of gradient clipping is standard practice for RNNs [REF], but to our knowledge, this is the first application of both global and localized clipping (and it is important to do both!)

Appears to be critical to limit the size of the overall norm of the gradient, particularly for some of the really long-delay tasks.

Critical to do a combination of both global and local norm clipping. Either alone is insufficient.

3.3 Architectural variations

3.3.1 Input masking ("Poor man's gate")

Because the recurrent layer uses no gating mechanism, it can sometimes be useful to have an extra feed forward hidden layer that can learn to selectively ignore certain inputs. In the pathological sequence problems, we have chosen to use ReLU units, because of their hard stop at zero. This is necessary for all of the "pathological" problems. "Poor man's gate" can only mask information that is immediately identifiable as not useful; makes use of no prior contextual information.

4 Experiments

Experimental settings, like learning rates, cooling schedules, hidden layer sizes, etc..

Table 1: Hooray for a random table

Case	Method#1	Method#2	Method#3
1	50	837	970
2	47	877	230
3	31	25	415
4	35	144	2356
5	45	300	556

All experiments use the following settings:

RMSprop*, with a learning rate of 0.001 Recurrent parameters are all initialized to 1.0 Feed forward parameters use Xavier's initialization [REF] Global gradient norm clipped at 30.0 Per-parameter gradients clipped at [-1.0, 1.0]

Initialize all recurrent factors to 1.0 Derivative clipping Weight initialization scheme *Try with mini-batches Hidden layer sizes Poor-man's gate sizes

Maybe make a table of these things for all the experiments? That would be pleasant to read

4.1 Pathological Sequence Problems

Maybe mention Hessian-Free learning here, even though it isn't really the same in terms of architecture. Reference the original LSTM papers that were the source of the pathological sequence problems. Mention again the use of the extra feed forward layer before the recurrent one for these.

Table 2: Example from bAbI problem set

Fact	Question	Answer
Mary moved to the bathroom. John went to the hallway.	Where is Mary?	bathroom
Daniel went back to the hallway. Sandra moved to the garden.	Where is Daniel?	hallway
John moved to the office. Sandra journeyed to the bathroom.	Where is Daniel?	hallway
Mary moved to the hallway. Daniel travelled to the office.	Where is Daniel?	office
John went back to the garden. John moved to the bedroom.	Where is Sandra?	bathroom

Reiterate that most of these toy problems emphasize a very small number of computations (applications of nonlinearity) spaced out over very long stretches of time.

4.1.1 Pathological Adding

(Turns out to be much faster to solve if you fix the recurrent weights to constant values of 1. Doing so still allows solutions to more complex tasks like bAbI, but with slightly worse scores.)

4.1.2 Pathological Multiplication

4.1.3 Pathological XOR

4.2 Sequential Parity

Mention evolino Can be represented by a very simple state machine (show diagram?), yet extremely difficult to train for conventional RNNs. Reiterate, long period of time plus lots of computations. LSTM and GRU fail spectacularly. *Maybe show diagram of hidden states through time?

4.3 Facebook bAbI Question-Answering

bAbI is a set of 20 synthetic NLP tasks put out by Facebook research (2015) which have been designed to test the ability to perform logical reasoning about text. Each task has been designed to require a particular set of capabilities to be solved; simple induction, deduction, remembering temporal ordering of facts, etc..

In the accompanying paper, Weston et al introduce Memory Networks in order to overcome many of the shortcomings of a plain LSTM network.

Additionally, it has been found that adding attentional mechanisms to input history has resulted in significantly improved performance.

ADD MORE STUFF ABOUT THAT HERE

Note that in this work we are not comparing Diagnet directly to any of the aforementioned "augmented" neural architectures. Instead, we demonstrate that Diagnet significantly outperforms LSTM and GRU on many of these 20 tasks, even despite using a much simpler and computationally efficient internal architecture.

As far as we are aware, these are the strongest results to date on the bAbI tasks for a "plain" neural neural architecture.

This suggests that Diagnet may represent

5 Discussion

5.1 Future Work

6 Conclusion

Here we outline the features, mechanisms, and benchmarking performance of Diagnet. Results were robust over a variety of tasks, most notably in bAbI question-answering tasks, where Diagnet's accuracy scores showed up to 36% percent improvement over existing architectures such as LSTM and GRU. While memory augmentation and attentional mechanisms offer clear advantages as architectural additions, we choose to focus on the core RNN functionality in this paper. Future work may incorporate attentional mechanisms, memory augmentation, and additional task spaces such as semantic embeddings, genetic sequencing, and computer vision. Diagnet's novel architecture offers great flexibility in hidden layering, and number of steps in various memory tasks, as well as potential for distributed computing. Due to the fact that Diagnet does not have gating mechanisms in its current iteration, it is largely input-driven, and tends to perform best in tasks where an output sequence is expected after arbitrary delays. The work presented here represents a first glance at Diagnet's mechanisms and potential, which the authors hope to expand in future work.

Acknowledgments

The authors would like to express their gratitude to their sponsor .

References

References follow the acknowledgments. Use unnumbered first-level heading for the references. Any choice of citation style is acceptable as long as you are consistent. It is permissible to reduce the font size to `small` (9 point) when listing the references. **Remember that you can use more than eight pages as long as the additional pages contain *only* cited references.**

[1] Alexander, J.A. & Mozer, M.C. (1995) Template-based algorithms for connectionist rule extraction. In G. Tesauro, D.S. Touretzky and T.K. Leen (eds.), *Advances in Neural Information Processing Systems 7*, pp. 609–616. Cambridge, MA: MIT Press.

[2] Bower, J.M. & Beeman, D. (1995) *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System.* New York: TELOS/Springer–Verlag.

[3] Hasselmo, M.E., Schnell, E. & Barkai, E. (1995) Dynamics of learning and recall at excitatory recurrent synapses and cholinergic modulation in rat hippocampal region CA3. *Journal of Neuroscience* **15**(7):5249-5262.