Implicitly-Defined Neural Networks for Sequence Labeling *

Michaeel Kazi, Brian Thompson
MIT Lincoln Laboratory
244 Wood St, Lexington, MA, 02420, USA
{first.last}@ll.mit.edu

Abstract

In this work, we propose a novel, implicitly-defined neural network architecture and describe a method to compute its components. The proposed architecture forgoes the causality assumption used to formulate recurrent neural networks and instead couples the hidden states of the network, allowing improvement on problems with complex, long-distance dependencies. Initial experiments demonstrate the new architecture outperforms both the Stanford Parser and baseline bidirectional networks on the Penn Treebank Part-of-Speech tagging task and a baseline bidirectional network on an additional artificial random biased walk task.

1 Introduction

Feedforward neural networks were designed to approximate and interpolate functions. Recurrent Neural Networks (RNNs) were developed to predict sequences. RNNs can be 'unwrapped' and thought of as very deep feedforward networks, with weights shared between each layer. Computation proceeds one step at a time, like the trajectory of an ordinary differential equation when solving an initial value problem. The path of an initial value problem depends only on the current state and the current value of the forcing function. In a RNN, the analogy is the current hidden state and the current input sequence. However, in certain applications in natural language processing, especially those with long-distance dependencies or where grammar matters, sequence prediction may be better thought of as a boundary value problem. Changing the value of the forcing function (analogously, of an input sequence element) at any point in the sequence will affect the values everywhere else. The bidirectional recurrent network (Schuster and Paliwal, 1997) attempts to addresses this problem by creating a network with two recurrent hidden states – one that progresses in the forward direction and one that progresses in the reverse. This allows information to flow in both directions, but each state can only consider information from one direction. In practice many algorithms require more than two passes through the data to determine an answer. We provide a novel mechanism that is able to process information in both directions, with the motivation being a program which iterates over itself until convergence.

1.1 Related Work

Bidirectional, long-distance dependencies in sequences have been an issue as long as there have been NLP tasks, and there are many approaches to dealing with them.

Hidden Markov models (HMMs) (Rabiner, 1989) have been used extensively for sequence-based tasks, but they rely on the Markov assumption – that a hidden variable changes its state based only on its current state and observables. In finding maximum likelihood state sequences, the Forward-Backward algorithm can take into account the entire set of observables, but the underlying model is still local.

In recent years, popularity of the Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) and variants such as the Gated Recurrent Unit (GRU) (Cho et al., 2014) has soared, as they enable RNNs to process long sequences without the problem of vanishing or exploding gradients (Pascanu et al., 2013). However, these models

*This work is sponsored by the Air Force Research Laboratory under Air Force contract FA-8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the authors and are not necessarily endorsed by the United States Government.

Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Short Papers), pages 172–177
Vancouver, Canada, July 30 - August 4, 2017.
https://doi.org/10.18653/v1/P17-2027

only allow for information/gradient information to flow in the forward direction.

The Bidirectional LSTM (b-LSTM) (Graves and Schmidhuber, 2005), a natural extension of (Schuster and Paliwal, 1997), incorporates past and future hidden states via two separate recurrent networks, allowing information/gradients to flow in both directions of a sequence. This is a very loose coupling, however.

In contrast to these methods, our work goes a step further, fully coupling the entire sequences of hidden states of an RNN. Our work is similar to (Finkel et al., 2005), which augments a CRF with long-distance constraints. However, our work differs in that we extend an RNN and uses Netwon-Krylov (Knoll and Keyes, 2004) instead of Gibbs Sampling.

2 The Implicit Neural Network (INN)

2.1 Traditional Recurrent Neural Networks

A typical recurrent neural network has a (possibly transformed) input sequence $[\xi_1, \xi_2, \ldots, \xi_n]$ and initial state h_s and iteratively produces future states:

$$\begin{aligned} h_1 &= f(\xi_1, h_s) \\ h_2 &= f(\xi_2, h_1) \\ &\ldots \\ h_n &= f(\xi_n, h_{n-1}) \end{aligned}$$

The LSTM, GRU, and related variants follow this formula, with different choices for the state transition function. Computation proceeds linearly, with each next state depending only on inputs and previously computed hidden states.

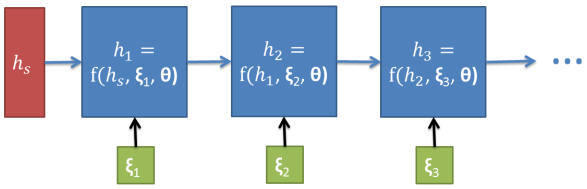

Figure 1: Traditional RNN structure.

2.2 Proposed Architecture

In this work, we relax this assumption by allowing $h_t = f(\xi_t, h_{t-1}, h_{t+1})$[1]. This leads to an implicit set of equations for the entire sequence of hidden states, which can be thought of as a single tensor H:

$$H = [h_1, h_2, \ldots, h_n]$$

This yields a system of nonlinear equations. This setup has the potential to arrive at nonlocal, whole sequence-dependent results. We also hope such a system is more 'stable', in the sense that the predicted sequence may drift less from the true meaning, since errors will not compound with each time step in the same way.

There are many potential ways to architect a neural network – in fact, this flexibility is one of deep learning's best features – but we restrict our discussion to the structure depicted in Figure 2. In this setup, we have the following variables:

data	X
labels	Y
parameters	θ

and functions:

input layer transformation	$\xi = g(\theta, X)$
implicit hidden layer def.	$H = F(\theta, \xi, H)$
loss function	$L = \ell(\theta, H, Y)$

Our implicit definition function, F, is made up of local state transitions and forms a system of nonlinear equations that require solving, denoting n as the length of the input sequence and h_s, h_e as boundary states:

$$\begin{aligned} h_1 &= f(h_s, h_2, \xi_1) \\ &\ldots \\ h_i &= f(h_{i-1}, h_{i+1}, \xi_i) \\ &\ldots \\ h_n &= f(h_{n-1}, h_e, \xi_n) \end{aligned}$$

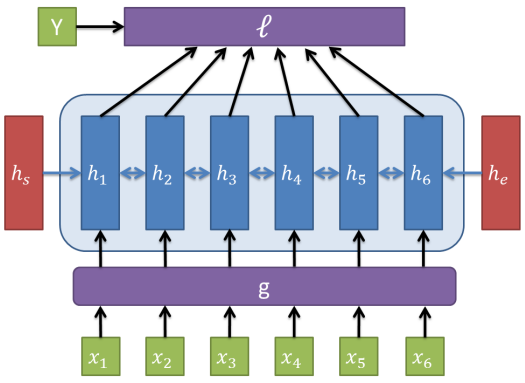

Figure 2: Proposed INN Architecture

[1] A wider stencil can also be used, e.g. $f(h_{t-2}, h_{t-1}, \ldots)$.

2.3 Computing the forward pass

To evaluate the network, we must solve the equation $H = F(H)$. We computed this via an approximate Newton solve, where we successively refine an approximation H_n of H:

$$H_{n+1} = H_n - (I - \nabla_H F)^{-1}(H_n - F(H_n))$$

Let k be the dimension of a single hidden state. $(I - \nabla_H F)$ is a sparse matrix, since $\nabla_H F$ is zero except for k pairs of $n \times n$ block matrices, corresponding to the influence of the left and right neighbors of each state.

Because of this sparsity, we can apply Krylov subspace methods (Knoll and Keyes, 2004), specifically the BiCG-STAB method (Van der Vorst, 1992), since the system is non-symmetric. This has the added advantage of only relying on matrix-vector multiplies of the gradient of F.

2.4 Gradients

In order to train the model, we perform stochastic gradient descent. We take the gradient of the loss function:

$$\nabla_\theta L = \nabla_\theta \ell + \nabla_H \ell \nabla_\theta H$$

The gradient of the hidden units with respect to the parameters can found via the implicit definition:

$$\begin{aligned} \nabla_\theta H &= \nabla_\theta F + \nabla_H F \nabla_\theta H + \nabla_\xi F \nabla_\theta \xi \\ &= (I - \nabla_H F)^{-1} (\nabla_\theta F + \nabla_\xi F \nabla_\theta \xi) \end{aligned}$$

where the factorization follows from the noting that

$$(I - \nabla_H F)\nabla_\theta H = \nabla_\theta F + \nabla_\xi F \nabla_\theta \xi.$$

The entire gradient is thus:

$$\begin{aligned} \nabla_\theta L =& \nabla_H \ell (I - \nabla_H F)^{-1} (\nabla_\theta F + \nabla_\xi F \nabla_\theta \xi) \\ &+ \nabla_\theta \ell \end{aligned} \tag{1}$$

Once again, the inverse of $I - \nabla_H F$ appears, and we can compute it via Krylov subspace methods. It is worth mentioning the technique of computing parameter updates by implicit differentiation and conjugate gradients have been applied before, in the context of energy minimization models in image labeling and denoising (Domke, 2012).

2.5 Transition Functions

Recall the original GRU equations (Cho et al., 2014), with slight notational modifications:

final h	$h_t = (1 - z_t)\hat{h}_t + z_t \tilde{h}_t$
candidate h	$\tilde{h}_t = \tanh(W x_t + U(r_t \hat{h}_t) + \tilde{b})$
update weight	$z_t = \sigma(W_z x_t + U_z \hat{h}_t + b_z)$
reset gate	$r_t = \sigma(W_r x_t + U_r \hat{h}_t + b_r)$

We make the following substitution for $\hat{h}_t$ (which was set to h_{t-1} in the original GRU definition):

state comb.	$\hat{h}_t = s h_{t-1} + (1 - s) h_{t+1}$
switch	$s = \frac{s_p}{s_p + s_n}$
prev. switch	$s_p = \sigma(W_p x_t + U_p h_{t-1} + b_p)$
next switch	$s_n = \sigma(W_n x_t + U_n h_{t+1} + b_n)$

(2)

This modification makes the architecture both implicit and bidirectional, since $\hat{h}_t$ is a linear combination of previous and future hidden states. The switch variable s is determined by a competition between two sigmoidal units s_p and s_n, representing the contributions of the previous and next hidden states, respectively.

2.6 Implementation Details

We implemented the implicit GRU structure using Theano (Bergstra et al., 2011). The product $\nabla_H F v$ for various v, required for the BiCG-STAB method, was computed via the `Rop` operator. In computing $\nabla_\theta L$ (Equation 1), we noted it is more efficient to compute $\nabla_H \ell (I - \nabla_H F)^{-1}$ first, and thus used the `Lop` operator.

All experiments used a batch size of 20. To batch solve the linear equations, we simply solved a single, very large block diagonal system of equations: each sequence in the batch was a single block matrix, and we input the encompassing matrix into our Theano BiCG solver. (In practice the block diagonal system is represented as a 3-tensor, but it is equivalent.) In this setup, each step does receive separate update directions, but one global step length. h_S and h_e were fixed at zero, but could be trained as parameters.

In solving multiple simultaneous systems of equations, we noted some elements converged significantly faster than others. For this reason, we found it helpful to run Newton's method from two separate initializations for each element in our batch, one selected randomly and the other set to a

"one-step" approximation: Hidden states of a traditional GRU were computed in both forward (h_i^f) and reverse (h_i^b) directions, and h_i was initialized to $f(h_{i-1}^f, h_{i+1}^b, \xi_i)$. If either of the two candidates converged, we took its value and stopped computing the other. We also limited both the number Newton iterations and BiCG-STAB iterations per Newton iteration to 40.

b	INN Error	b-LSTM Error
2.0	0.0226	**0.0210**
1.0	**0.0518**	0.0589
0.75	**0.0782**	0.0879
0.5	**0.119**	0.132
0.25	**0.189**	0.205

Table 1: Biased walk classification performance.

3 Experiments

3.1 Biased random walks

We developed an artificial task with bidirectional sequence-level dependencies to explore the performance of our model. Our task was to find the point at which a random walk, in the spirit of the Wiener Process (Durrett, 2010), changes from a zero to nonzero mean. We trained a network to predict when the walk is no longer unbiased. We generated algorithmic data for this problem, the specifics of which are as follows: First, we chose an integer interval length N uniformly in the range 1 to 40. Then, we chose a (continuous) time $t' \in [0, N)$, and a direction $v \in \mathcal{R}^d$. We produced the input sequence $x_i \in \mathcal{R}^d$, setting $x_0 = 0$ and iteratively computing $x_{i+1} = x_i + \mathcal{N}(0, 1)$. After time t, a bias term of $b \cdot v$ was added at each time step ($b \cdot v \cdot (t' - t)$) for the first time step greater than t'. b is a global scalar parameter. The network was fed in these elements, and asked to predict $y = 0$ for times $t \leq t'$ and $y = 1$ for times $t > t'$.

For each architecture, ξ was simply the unmodified input vectors, zero-padded to the embedding dimension size. The output was a simple binary logistic regression. We produced 50,000 random training examples, 2500 random validation examples, and 5000 random test examples. The implicit algorithm used a hidden dimension of 200, and the b-LSTM had an embedding dimension ranging from 100 to 1000. b-LSTM dimension of 300 was the point where the total number of parameters were roughly equal.

The results are shown in Table 1. The b-LSTM scores reported are the maximum over sweeps from 100 to 1500 hidden dimension size. The INN outperforms the best b-LSTM in the more challenging cases where the bias size b is small.

3.2 Part-of-speech tagging

We next applied our model to a real-world problem. Part-of-speech tagging fits naturally in the sequence labeling framework, and has the advantage of a standard dataset that we can use to compare our network with other techniques. To train a part-of-speech tagger, we simply let L be a softmax layer transforming each hidden unit output into a part of speech tag. Our input encoding ξ, is a concatenation of three sets of features, adapted from (Huang et al., 2015): first, word vectors for 39,000 case-insensitive vocabulary words; second, six additional 'word vector' components indicating the presence of the top-2000 most common prefixes and suffixes of words, for affix lengths 2 to 4; and finally, eight other binary features to indicate the presence of numbers, symbols, punctuation, and more rich case data.

We trained the Part of Speech (POS) tagger on the Penn Treebank Wall Street Journal corpus (Marcus et al., 1993), blocks 0-18, validated on 19-21, and tested on 22-24, per convention. Training was done using stochastic gradient descent, with an initial learning rate of 0.5. The learning rate was halved if validation perplexity increased. Word vectors were of dimension 320, prefix and suffix vectors were of dimension 20. Hidden unit size was equal to feature input size, so in this case, 448.

As shown in Table 2, the INN outperformed baseline GRU, bidirectional GRU, LSTM, and b-LSTM networks, all with 628-dimensional hidden layers (1256 for the bidirectional architectures), The INN also outperforms the Stanford Part-of-Speech tagger (Toutanova et al., 2003) (model `wsj-0-18-bidirectional-distsim.tagger` from 10-31-2016). Note that performance gains past approximately 97% are difficult due to errors/inconsistencies in the dataset, ambiguity, and complex linguistic constructions including dependencies across sentence boundaries (Manning, 2011).

Architecture	WSJ Accuracy
GRU	96.43
LSTM	96.47
Bidirectional GRU	97.28
b-LSTM	97.25
INN	**97.37**
Stanford POS Tagger	97.33

Table 2: Tagging performance relative to recurrent architectures and Stanford POS Tagger.

4 Time Complexity

The implicit experiments in this paper took approximately 3-5 days to run on a single Tesla K40, while the explicit experiments took approximately 1-3 hours. Running time of the solver is approximately $n_n \times n_b \times t_b$ where n_n is the number of Newton iterations, n_b is the number of BiCG-STAB iterations, and t_b is the time for a single BiCG-STAB iteration. t_b is proportional to the number of non-zero entries in the matrix (Van der Vorst, 1992), in our case $n(2k^2 + 1)$. Newton's method has second order convergence (Isaacson and Keller, 1994), and while the specific bound depends on the norm of $(I - \nabla_H F)^{-1}$ and the norm of its derivatives, convergence is well-behaved. For n_b, however, we are not aware of a bound. For symmetric matrices, the Conjugate Gradient method is known to take $O(\sqrt{\kappa})$ iterations (Shewchuk et al., 1994), where κ is the condition number of the matrix. However, our matrix is nonsymmetric, and we expect κ to vary from problem to problem. Because of this, we empirically estimated the correlation between sequence length and total time to compute a batch of 20 hidden layer states.

For the random walk experiment with $b = 0.5$, we found the the average run time for a given sequence length to be approximately $0.17n^{1.8}$, with $r^2 = 0.994$. Note that the exponent would have been larger had we not truncated the number of BiCG-STAB iterations to 40, as the inner iteration frequently hit this limit for larger n. However, the average number of Newton iterations did not go above 10, indicating that exiting early from the BiCG-STAB loop did not prevent the Newton solver from converging. Run times for the other random walk experiments were very similar, indicating run time does not depend on b; However, for the POS task runtime was $0.29n^{1.3}$, with $r^2 = 0.910$.

5 Conclusion and Future Work

We have introduced a novel, implicitly defined neural network architecture based on the GRU and shown that it outperforms a b-LSTM on an artificial random walk task and slightly outperforms both the Stanford Parser and a baseline bidirectional network on the Penn Treebank Part-of-Speech tagging task.

In future work, we intend to consider implicit variations of other architectures, such as the LSTM, as well as additional, more challenging, and/or data-rich applications. We also plan to explore ways to speed up the computation of $(I-\nabla_H F)^{-1}$. Potential speedups include approximating the hidden state values by reducing the number of Newton and/or BiCG-STAB iterations, using cached previous solutions as initial values, and modifying the gradient update strategy to keep the batch full at every Newton iteration.

6 Acknowledgements

This work would not be possible without the support and funding of the Air Force Research Laboratory. We also acknowledge Nick Malyska, Elizabeth Salesky, and Jonathan Taylor at MIT Lincoln Lab for interesting technical discussions related to this work.

Cleared for Public Release on 29 Jul 2016. Originator reference number: RH-16-115722. Case Number: 88ABW-2016-3809.

References

James Bergstra, Frédéric Bastien, Olivier Breuleux, Pascal Lamblin, Razvan Pascanu, Olivier Delalleau, Guillaume Desjardins, David Warde-Farley, Ian Goodfellow, Arnaud Bergeron, et al. 2011. Theano: Deep learning on gpus with python. In *NIPS 2011, BigLearning Workshop, Granada, Spain*.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder–decoder approaches. *Syntax, Semantics and Structure in Statistical Translation* page 103.

Justin Domke. 2012. Generic methods for optimization-based modeling. In *AISTATS*. volume 22, pages 318–326.

Richard Durrett. 2010. *Probability : theory and examples*. Cambridge University Press, Cambridge New York.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, pages 363–370.

Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks* 18(5):602–610.

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

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991* .

Eugene Isaacson and Herbert Bishop Keller. 1994. *Analysis of numerical methods*. Courier Corporation, New York.

Dana A Knoll and David E Keyes. 2004. Jacobian-free newton–krylov methods: a survey of approaches and applications. *Journal of Computational Physics* 193(2):357–397.

Christopher D Manning. 2011. Part-of-speech tagging from 97% to 100%: is it time for some linguistics? In *International Conference on Intelligent Text Processing and Computational Linguistics*. Springer, pages 171–189.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Computational linguistics* 19(2):313–330.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. *ICML (3)* 28:1310–1318.

Lawrence R Rabiner. 1989. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE* 77(2):257–286.

Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on* 45(11):2673–2681.

Jonathan Richard Shewchuk et al. 1994. An introduction to the conjugate gradient method without the agonizing pain.

Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*. Association for Computational Linguistics, pages 173–180.

Henk A Van der Vorst. 1992. Bi-cgstab: A fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems. *SIAM Journal on scientific and Statistical Computing* 13(2):631–644.