Longer RNNs

Ashish Bora Aishwarya Padmakumar Akanksha Saran

Abstract

For problems with long range dependencies, training of Recurrent Neural Networks (RNNs) faces limitations because of vanishing or exploding gradients. In this work, we introduce a new training method and a new recurrent architecture using residual connections, both aimed at increasing the range of dependencies that can be modeled by RNNs. We demonstrate the effectiveness of our approach by showing faster convergence on two toy tasks involving long range temporal dependencies, and improved performance on a character level language modeling task. Further, we show visualizations which highlight the improvements in gradient propagation through the network.

1. Introduction

Recurrent Neural Networks (RNNs) are powerful models for learning in tasks involving sequential input/output, possibly of variable length. There have been several successful applications of RNNs to various domains such as machine translation [21], image and video captioning [25, 23], and speech recognition [3]. Several state of the art methods use RNNs [13, 18, 19, 7].

Despite tremendous success, learning long range dependencies with vanilla RNNs is difficult [5]. A well known reason for this problem is that gradient based training algorithms suffer from the problem of vanishing or exploding gradients [1, 16].

Many solutions have been proposed to handle this problem, the most successful of which has been Long Short Term Memory (LSTM) units [6]. More recently, RNNs have been shown to work on moderately long sequences by using ReLU activation and initializing the hidden to hidden connections with identity matrices (iRNN [11]). iRNNs have the advantage of a smaller number of parameters and simpler computations compared to LSTMs. The iRNN model obtains superior results as compared to LSTMs on various tasks as shown by Le *et al.* [11]. iRNNs have been subsequently improved by Krueger *et al.* [10]. Despite many advances, these models are still unable to learn well on a simple addition task (Section 5.1) for sequences of length larger than 400 which highlights their limitations.

Towards learning long range dependencies with recurrent networks, our contributions are two-fold. First, we propose a new training algorithm that relaxes tying of RNN parameters across time (Section 2). This approach is generic and applicable to a wide variety of recurrent cells. Second, we propose a new recurrent architecture using residual connections (Section 3) which is well suited to handle long term dependencies.

For our experiments we use two toy tasks: addition and multiplication (Sections 5.1, 5.2) and a language modeling task (Section 5.3). Experiments with our training algorithm give us insights into the training of iRNNs and enable us to learn with longer sequences on the addition task as compared to prior work [11, 10]. With our residual recurrent architecture, we demonstrate even faster convergence for learning addition and multiplication tasks on sequences as long as 600 time steps. We also show improved performance on character level language modeling of the Penn Treebank dataset [14] on sequences of length 50. Gradient visualizations give us insights into the training procedure and expose some limitations of the iRNN model.

2. Very Long RNNs

As noted above, exploding and vanishing gradients are a significant hurdle for gradient based learning in recurrent architectures. We observe that this happens due to repeated application of the same transformation at each time step. To a first order approximation, if the transformation per step is multiplication by a matrix (say A) at each time step, then the hidden state at time t is $A^t h_0$, where h_0 is the initial hidden state. If the eigenvalues of the matrix are larger than 1, the activations explode and if they are less than 1, they vanish. This is particularly problematic in RNNs that apply sigmoid or tanh non-linearities over the activations as even moderately large or small activations can lead to saturation of the non-linearity and consequently, near zero gradients. If ReLUs are chosen as the non-linearity instead, it does not saturate but can instead be highly unstable during training with a high learning rate, or train very slowly when a low learning rate is used.

iRNNs [11] propose to solve this problem by initializing the network weights such that the matrix A is identity. This prevents activations from vanishing or exploding at the start of training. We observe that even though this is a very good initialization, it does not solve the problem completely. Indeed, as we train the model, the transformation will change away from identity and the network faces vanishing/exploding gradients problem. In 6.9 we show several visualizations that agree with this intuition.

The chief culprit for these problems is that we apply the *same* transformation at every time step. This leads to extreme sensitivity towards changes in the per step transformation. But using the same transformation need not be true anytime during learning at all, except at test time, to allow the trained network to be relatively small in size and also guard against overfitting.

Based on this, we propose a new training approach to handle the vanishing or exploding gradients problem. During training, instead of forcing all RNN weights to be the same at each time step, we allow them to be different. Then, as the network learns, we slowly encourage the weights to be closer to each other by introducing a penalty term whose strength grows as we train. This method allows for temporal decoupling of learnable parameters which might lead to better gradient propagation. We call iRNN models trained with this method 'Very Long RNN' or 'VL-iRNN'.

Formally, let RNN_θ be an RNN cell parameterized by θ. Let us assume that we apply it to a length T sequence $\{x_i\}_{i=1}^T$ and the loss as a result is $L(\theta)$. We use a new set of RNN parameters θ_t for each time step $t \in [T]$. Then the RNN loss is a function of all the parameters, i.e. $L((\theta_i)_{i=1}^T)$. We add a penalty term

$$P((\theta_i)_{i=1}^T) = \rho \sum_{i=1}^{T-1} \|\theta_i - \theta_{i+1}\|^2$$

where ρ is a penalty multiplier. We then do gradient based learning on the sum of the RNN loss and the penalty term. We initialize ρ with a very small value and gradually increase it. This allows for decoupled learning initially, and serves to bring the matrices together to each other near the end of training.

One of the drawbacks of this approach is that while training we need to store one set of parameters for each time step. This can take up large amount of memory, especially if the sequences are very long. To ameliorate this, we use a modified scheme where the parameters for first several steps are the same, followed by a new set of parameters for the next few steps and so on. By bucketing the time steps into consecutive groups, we can keep the memory requirement under control. We call each of these consecutive blocks a scope.

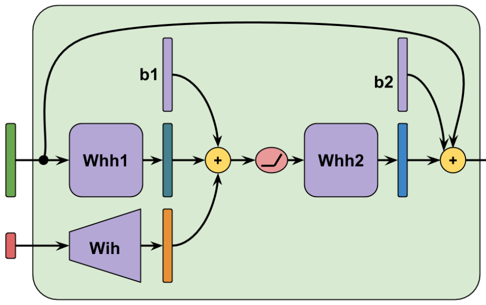


Figure 1: ResRNN cell. The blocks with purple color have learnable parameters.

Similar idea of decoupling parameters is well known in convex optimization literature. However the applications there, like parallelizability in the dual ascent algorithm, are different from our setting. Our contribution is aimed at the application of variable splitting to improve gradient propagation while training RNNs.

Similar to the interior point methods in linear programming [8], we use an exponential schedule for increasing the penalty multiplier.

3. Residual RNN

Residual connections were recently introduced in He *et al.*, 2015 [4]. The residual connections allow for better propagation of gradients, thus allowing fast training of convolutional networks with as many as 1000 layers. Architectures using residual connections achieve state of the art results in many vision tasks [4, 22]. Inspired by their recent success, we introduce a residual RNN (resRNN) architecture to model long range dependencies. Note that unlike standard residual networks, the weights of RNNs at each time step are tied to each other.

A diagram of the residual cell is shown in Fig. 1. We use a combination layer (W_{hh2}, b_2) before adding the residual connection. This is essential for the non-residual RNN part to be able to learn to subtract the input given by the residual connection, if necessary.

Our initialization for parameters in this model is inspired by the iRNN initialization. To make the hidden to hidden transitions be identity in the absence of any inputs, we initialize W_{hh1}, b_1, W_{hh2} and b_2 to have all entries equal to zero.

4. Related Work

The most commonly used modifications of RNNs for handling long-range dependencies is the Long Short Term

Memory (LSTM) [6] or Gated Recurrent Units (GRU) [2]. These significantly increase the number of parameters of the network and iRNN [11] is an initialization technique that provides comparable performance with LSTMs without this drastic increase in the number of parameters as shown by Le *et al.* [11].

An extension of iRNNs to prevent exploding gradients outside the training horizon is the work of Krueger *et al.*, 2015 [10], which penalizes the norm of the hidden state vectors. This is similar to the idea of penalty in our proposed VL-iRNN algorithm 2. However, we modify the learning procedure rather than imposing a constraint on the model itself.

There have also been some attempts to incorporate residual connections into recurrent architectures. Pradhan *et al.* [17] add residual connections to LSTMs and demonstrate improved performance on a sentiment analysis task. In contrast, our proposed resRNN incorporates residual connections into a vanilla RNN, which also keeps the number of parameters much lower than that required by an LSTM. Another model more similar to ours is that of Wang *et al.* [24]. In comparison to their work, we use a simpler 2-step transform at each recurrent step, and ReLU as a non-linearity instead of tanh.

Liao *et al.*, 2016 [12] demonstrate formally that a deep residual network with weight sharing is formally equivalent to a shallow RNN, and they propose a model that generalizes the two. However it is formulated in terms of dynamical systems and it is not clear whether the model is capable of handling longer range dependencies. It is proposed as a model of the visual cortex and experiments only consider tasks typically modeled using convolutional networks, not sequence tasks.

Another related model is the recurrent highway network by Zilly *et al.*, 2016 [26]. These networks are both deep in terms of unrolling in time as well as in space via multiple highway layers in the LSTM cell, extending the LSTM architecture to larger step-to-step transition depths. However, their goal is to model more complex state transitions in an LSTM cell without further increasing the difficulty of training. However, they do not focus on modeling long range dependencies and the maximum recurrence depth used in their experiments is 10. Our goal is different in that we aim to model long sequences and use fewer parameters than an LSTM.

5. Tasks

We employ three tasks to test VLiRNN, resRNN and compare them to iRNN. We describe each task in detail below.

5.1. Addition

The addition task is a toy problem designed to test the ability of recurrent networks to handle long-range dependencies [11]. It is a regression task over a 2-dimensional input sequence. At each time step, the first dimension of the input is a signal which is drawn uniformly at random from $[0, 1]$. The second dimension is a binary mask, which is 0 everywhere, except at exactly two time steps in the sequence chosen at random where it is 1. The recurrent network reads the entire sequence and must then predict the sum of the two signals corresponding to time steps when the mask has value 1. The loss function used is the mean squared error between the predicted and true targets.

The simplest baseline is to predict the target to be the mean of the target distribution regardless of the inputs. This results in a mean squared error equal to the variance of the target distribution, which is about 0.1767. The goal is to train a model that obtains a mean squared error considerably lower than this estimate on an unseen test set.

For a given sequence length, training and test sets with input sequences are generated beforehand and the same training and test sets are used for all models. By default we use a training set of 100000 sequences and a test set of 10000 sequences.

5.2. Multiplication

The multiplication task is another toy task similar to the addition task. The input sequence is similar to the addition task, except that the signal is drawn from a uniform distribution over $[0, 2]$. The target to be predicted is the product of the two signals corresponding to time steps with a mask of 1, instead of the sum. The loss function is again the mean squared error between the predicted and true targets. Training and test sets are created in a manner similar to the addition task.

Always predicting the mean of the target distribution achieves a mean squared error equal to the variance of the target distribution, which is 0.778. A good model should obtain a mean squared error considerably lower than this value.

5.3. Language Modeling

We also evaluate our models on the task of character-level language modeling to demonstrate the applicability of our models to real-world tasks [10].

Character level language models can be used to model unseen words in speech recognition, language understanding or keyword spotting tasks. Further, the standard smoothing techniques used in traditional n-gram language models work poorly for these tasks, increasing the need for good neural language models [15].

We use the Penn Treebank [14] dataset to evaluate the language modeling task. We use the preprocessed character level text from Mikolov *et al.*, 2012 [15] and the same train and test splits. We note that while preprocessing, spaces between words and sentence boundaries are marked by two special characters. In total, this gives us a character level vocabulary of size 50. At each time step, the input to the recurrent network is the character at the current time step, one hot encoded using the character vocabulary. The output is a prediction of the character at the next time step, in the form of a probability distribution over the characters in the vocabulary.

Similar to [10], the text is split into sequences of a fixed length both during training and testing. The loss function used during training is the cross-entropy with respect to the true next character at each time step. The task is evaluated in terms of cross entropy loss per character over the test set.

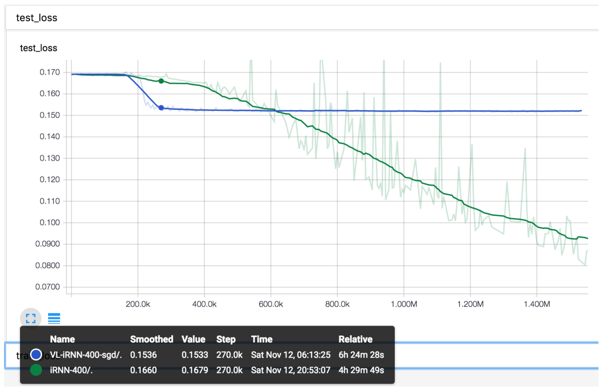
6. Experiments

The following experiments compare our proposed models with iRNN on the tasks described in section 5. We do not compare with LSTMs because Le *et al.*, 2015 [11] demonstrate that iRNNs are comparable with the standard implementation of LSTMs on similar tasks. Unless mentioned otherwise, experiments use a hidden state vector of size 100, initialization as in Le *et al.*, 2015 [11], a learning rate of 0.001, gradient clipping of 0.1, train batch size of 16 and a test batch size of 20. We experimented with different values for learning rate and gradient clipping but these produced the best results in most cases.

6.1. Addition task - iRNN vs VL-iRNN

We compare VL-iRNN to iRNN on the addition task with sequence length 400. VL-iRNN and iRNN both are trained using SGD as solvers in this experiment, following the implementation details of [11]. For VL-iRNN, we divide the time steps into 10 scopes, each of length 40. The test loss can be seen in Fig. 2. We tried several learning rates for each model, but only the best models are shown.

We observe that VL-iRNN starts to learn faster than iRNN, but saturates at a high error. The iRNN is initially very slow to learn but eventually obtains a much lower test loss. However, the test performance of iRNN shows a considerable amount of variance. We were unable to increase the speed of learning for iRNN by using a larger learning rate because it quickly lead to instability and divergence. Using lower learning rates would result in less variance in the test loss and also make learning slower.



	Name	Smoothed	Value	Step	Time	Relative
●	VL-iRNN-400-sgd/.	0.1536	0.1533	270.0k	Sat Nov 12, 06:13:25	6h 24m 28s
●	iRNN-400/.	0.1660	0.1679	270.0k	Sat Nov 12, 20:53:07	4h 29m 49s

Figure 2: Test loss for VL-iRNN and iRNN on the addition task for sequences of length 400. Both VL-iRNN and iRNN use SGD as the solver in this experiment.

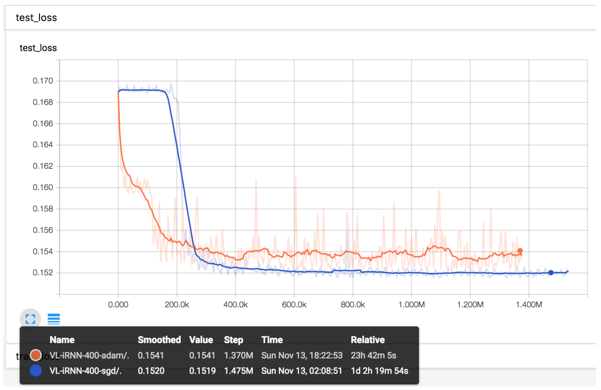
6.2. VL-iRNN - Different learning algorithms

There can be several reasons for why VL-iRNN saturates. To eliminate some of these, we note that the test set error of any learning algorithm can be decomposed into three components -

1. Sampling error : The training set is finite and cannot represent the underlying distribution fully
2. Bias : The hypothesis class of the learning algorithm is too limited
3. Optimization error : We are not able to find good functions within our hypothesis class due to limitations of the optimization procedure.

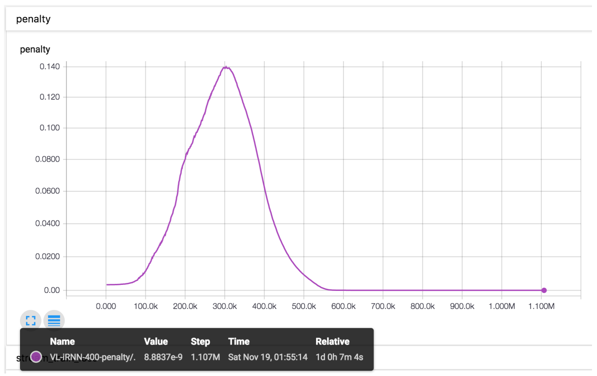
Sampling error can be ruled out because iRNNs do learn with the same training set (before becoming unstable). Bias cannot be the explanation either since VL-iRNN has a lower bias than iRNN. We also verified that the VL-iRNN is not overfitting - train and test losses are comparable and show the same trend over time. This leaves us with optimization error to be the most likely explanation.

In this experiment, we tried several learning algorithms (each with a range of hyperparameters) to see if we can overcome the difficulty in optimization. In Fig.5, we show the results with Stochastic Gradient Descent (SGD), Adam [9], and SGD with momentum [20]. We see that SGD and Adam (Fig. 3 shows a closer view of Fig. 5 for these solvers) both saturate to about the same test loss. SGD with momentum (see VL-iRNN-400-momentum in Fig. 5) how-



	Name	Smoothed	Value	Step	Time	Relative
●	VL-iRNN-400-adam/.	0.1541	0.1541	1.370M	Sun Nov 13, 18:22:53	23h 42m 5s
●	VL-iRNN-400-sgd/.	0.1520	0.1519	1.475M	Sun Nov 13, 02:08:51	1d 2h 19m 54s

Figure 3: VL-iRNN with adam and SGD solvers tested on the addition task for sequences of length 400.



	Name	Value	Step	Time	Relative
●	VL-iRNN-400-penalty/.	8.8837e-9	1.107M	Sat Nov 19, 01:55:14	1d 0h 7m 4s

Figure 4: Evolution of the penalty value (without the multiplier) for VL-iRNN with penalty. The experiment is performed on the addition task for sequences of length 400.

ever enables VL-iRNNs to learn very well. We also note that in 1.8 million iterations, VL-iRNN with momentum converges to a lower test loss than the lowest achieved by iRNN as shown by Le *et al*. [11], even when trained for 9 million iterations.

6.3. VL-iRNN with penalty

The previous experiment shows that the VL-iRNN model reliably and quickly learns the addition task on sequences of length 400 with 10 scopes. It is however not ideal since we have to use 10 times more parameters than the iRNN model. Thus, as described in Section 2, we introduce a penalty term to bring the matrices closer to each other, so that the trained network can be made to use only a single scope.

The penalty multiplier starts out at 2×10^{-6} and is multiplied by 1.001 every 37 training iterations. These numbers are chosen such that the penalty multiplier is about 10^6 at 1 million iterations.

In Fig. 5, we show the test loss for this method (see VL-iRNN-400-penalty in Fig. 5) as compared to others (see VL-iRNN-400-adam, VL-iRNN-400-momentum and VL-iRNN-400-sgd). We observe that even with the penalty term, using SGD with momentum, the VL-iRNN attains a test loss similar to when there is no penalty between scopes.

In Fig. 4 we show the evolution of the penalty value (without the multiplier). As expected we see that the penalty grows initially since the multiplier is small. After a point, the multiplier becomes large enough that the penalty starts to reduce. Finally, the penalty becomes almost zero indicating that all consecutive parameters sets are very close to each other. This demonstrates that the penalty method is effective in bringing parameters across different scopes close to each other.

6.4. iRNN with Momentum

Comparing Fig. 4 and Fig. 5 (VL-iRNN-400-penalty), we see that rapid learning of the VL-iRNN starts only when the penalty term is near zero. At this point, parameters across different scopes are very close to each other and thus the VL-iRNN is essentially an iRNN. Since we are able to learn with parameters across scopes being almost identical, it is important to examine whether the VL-iRNN obtains better results than the iRNN due to the decoupling of parameters, or because we use SGD with momentum for optimization instead of vanilla SGD.

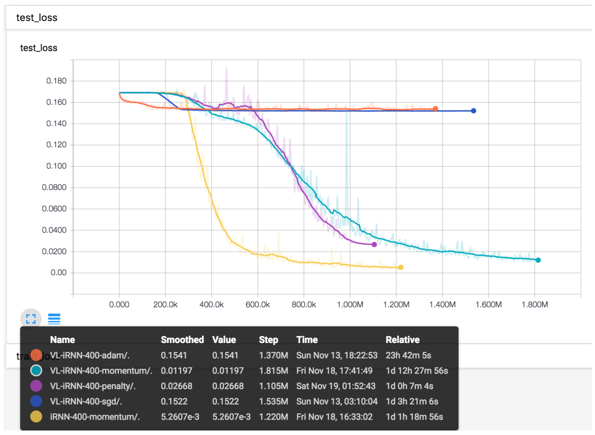
To answer this, we attempted training iRNNs using SGD with momentum. The results are shown in Fig. 5 (see iRNN-400-momentum). We see that this achieves very good performance as compared to other models – it converges faster and to a better value.

6.5. Limits of iRNN

Since iRNN with momentum works the best, in this experiment we test the limits of this model to see the range of dependencies it can model. We use the same task, i.e. addition, but use longer sequences of length 500 and 600.

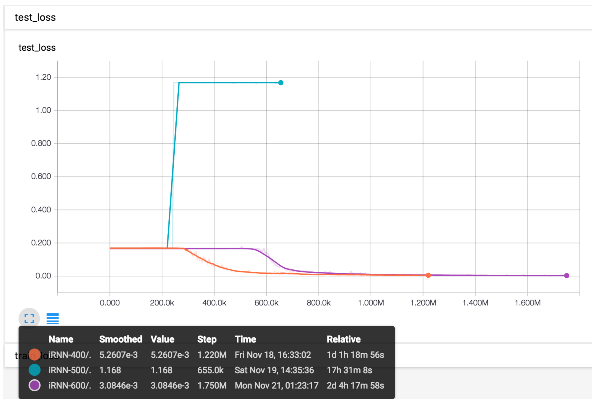
The results are shown in Fig. 6. We see that the model converges for sequences of length 600, but diverges for 500. The learning is already extremely slow; the model doesn't show any drop in the error till about 550k iterations for length 600. Thus, while it is possible that decreasing the learning rate will enable learning on length 500, it will make learning even slower which is undesirable.

This experiment also shows that iRNNs trained using SGD with momentum is still a very unstable algorithm.



	Name	Smoothed	Value	Step	Time	Relative
●	VL-iRNN-400-adam/.	0.1541	0.1541	1.370M	Sun Nov 13, 18:22:53	23h 42m 5s
●	VL-iRNN-400-momentum/.	0.01197	0.01197	1.815M	Fri Nov 18, 17:41:49	1d 12h 27m 56s
●	VL-iRNN-400-penalty/.	0.02668	0.02668	1.105M	Sat Nov 19, 01:52:43	1d 0h 7m 4s
●	VL-iRNN-400-sgd/.	0.1522	0.1522	1.535M	Sun Nov 13, 03:10:04	1d 3h 21m 6s
●	iRNN-400-momentum/.	5.2607e-3	5.2607e-3	1.220M	Fri Nov 18, 16:33:02	1d 1h 18m 56s

Figure 5: Test loss comparison for VL-iRNN with different solvers and iRNN with SGD and momentum. The experiment is performed for the addition task on sequences of length 400.



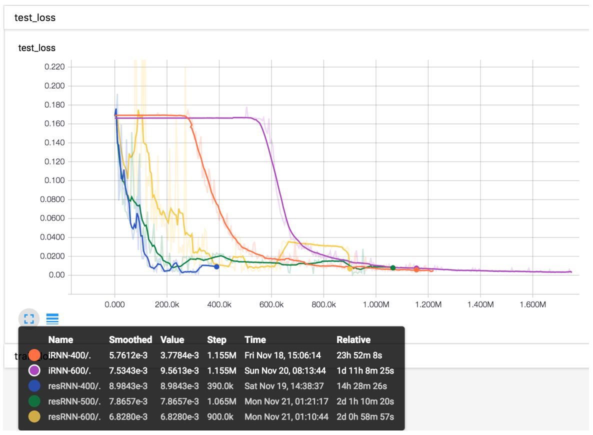
	Name	Smoothed	Value	Step	Time	Relative
●	iRNN-400/.	5.2607e-3	5.2607e-3	1.220M	Fri Nov 18, 16:33:02	1d 1h 18m 56s
●	iRNN-500/.	1.168	1.168	655.0k	Sat Nov 19, 14:35:36	17h 31m 8s
●	iRNN-600/.	3.0846e-3	3.0846e-3	1.750M	Mon Nov 21, 01:23:17	2d 4h 17m 58s

Figure 6: Performance of iRNNs trained using SGD with momentum on the addition task for sequences of length 500 and 600.

6.6. resRNN

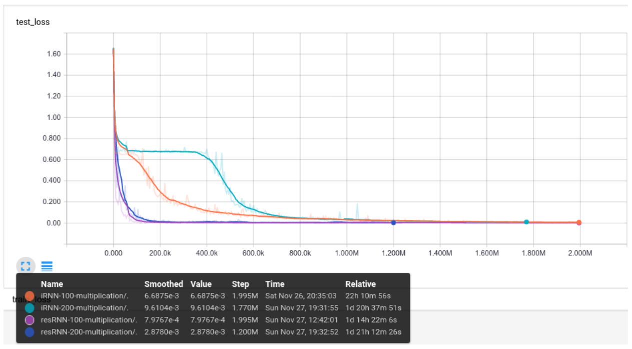
Since iRNNs are unstable and VL-iRNNs reduce to iRNNs when the penalty term is used, another alternative is still necessary for modeling long sequences. In this experiment, we explore the potential of our other proposed model - resRNNs (Section 3). Fig. 7 shows the test loss of the resRNN on the addition task using sequences of length 400.

As can be seen in the figure, resRNN learns faster than iRNN. In fact, resRNNs learn the addition task very fast for sequences of length 500 and 600 as well, compared to iRNNs. Though there are minor oscillations in the test loss as the sequences get longer, resRNNs outperform iRNNs for these long sequences.



	Name	Smoothed	Value	Step	Time	Relative
●	iRNN-400/.	5.7612e-3	3.7784e-3	1.155M	Fri Nov 18, 15:06:14	23h 52m 8s
●	iRNN-600/.	7.5343e-3	9.5613e-3	1.155M	Sun Nov 20, 08:13:44	1d 11h 8m 25s
●	resRNN-400/.	8.9843e-3	8.9843e-3	390.0k	Sat Nov 19, 14:38:37	14h 28m 26s
●	resRNN-500/.	7.8657e-3	7.8657e-3	1.065M	Mon Nov 21, 01:21:17	2d 1h 10m 20s
●	resRNN-600/.	6.8280e-3	6.8280e-3	900.0k	Mon Nov 21, 01:10:44	2d 0h 58m 57s

Figure 7: Test performance of resRNN and iRNN for the addition task on sequences of length 400. Both use SGD with momentum as the solver for this experiment.



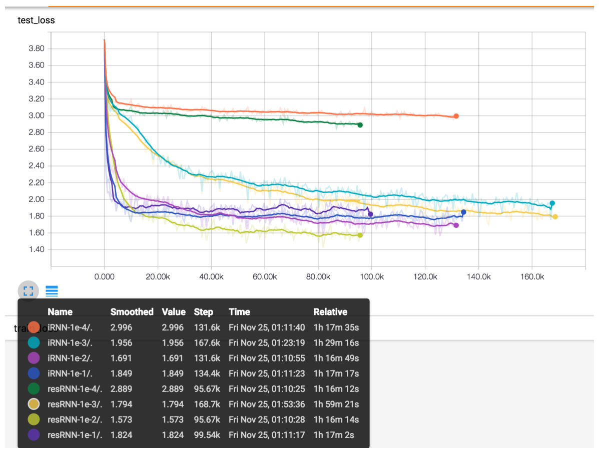
	Name	Smoothed	Value	Step	Time	Relative
●	iRNN-100-multiplication/.	6.6875e-3	6.6875e-3	1.995M	Sat Nov 26, 20:35:03	22h 10m 56s
●	iRNN-200-multiplication/.	9.6104e-3	9.6104e-3	1.770M	Sun Nov 27, 19:31:55	1d 20h 37m 51s
●	resRNN-100-multiplication/.	7.9767e-4	7.9767e-4	1.995M	Sun Nov 27, 12:42:01	1d 14h 22m 6s
●	resRNN-200-multiplication/.	2.8780e-3	2.8780e-3	1.200M	Sun Nov 27, 19:32:52	1d 21h 12m 26s

Figure 8: Test performance of resRNN and iRNN on the multiplication task with sequences of length 100 and 200.

6.7. Multiplication Task

One can argue that the capability of resRNNs to outperform iRNNs on the addition task is inherent to the architecture of the residual cell (Section 3). Since the residual cell (Fig. 1) has an in-built addition operator directly using the input, the performance of resRNN may be attributed to the task being almost encoded in the architecture. To decouple the potential of resRNNs in learning longer sequences from the addition task, we also test its performance on the multiplication task as described in Section 5.2.

We can see in Fig. 8 that resRNNs learn multiplication faster than iRNNs for sequences of length 100 and 200. We tried experimenting with sequences of longer length (400, 500) but did not find any hyperparameters leading to stable performance for either resRNNs or iRNNs. A full fledged grid search over hyperparameter values for longer sequences remains to be done and would be part of future investigations.



	Name	Smoothed	Value	Step	Time	Relative
●	iRNN-1e-4/.	2.996	2.996	131.6k	Fri Nov 25, 01:11:40	1h 17m 35s
●	iRNN-1e-3/.	1.956	1.956	167.6k	Fri Nov 25, 01:23:19	1h 29m 16s
●	iRNN-1e-2/.	1.691	1.691	131.6k	Fri Nov 25, 01:10:55	1h 16m 49s
●	iRNN-1e-1/.	1.849	1.849	134.4k	Fri Nov 25, 01:11:23	1h 17m 17s
●	resRNN-1e-4/.	2.889	2.889	95.67k	Fri Nov 25, 01:10:25	1h 16m 12s
●	resRNN-1e-3/.	1.794	1.794	168.7k	Fri Nov 25, 01:53:36	1h 59m 21s
●	resRNN-1e-2/.	1.573	1.573	95.67k	Fri Nov 25, 01:10:28	1h 16m 14s
●	resRNN-1e-1/.	1.824	1.824	99.54k	Fri Nov 25, 01:11:17	1h 17m 2s

Figure 9: Character level language modeling for sequences of length 50 from the Penn Tree Bank dataset [14].

6.8. Language Modeling

We also compare resRNNs and iRNNs on a real-world application - character level language modeling (Section 5.3). The test loss over different learning rates can be seen in Fig. 9 for a fixed sequence length of 50, with 100 hidden units in each architecture. We again see that the test loss of resRNNs is consistently lower than iRNNs. Hence we find that resRNNs with momentum are a better model for learning longer sequences than iRNNs with momentum.

6.9. Gradient Analysis

From the various learning curves, we see that the test loss for iRNNs stays at a high value for a very long period at the beginning of training. This is seen consistently for many training algorithms, hyperparameter settings and sequence lengths. Here, we present visualizations that can help us understand this phenomenon.

We aim to visualize the gradient propagation in the network. Even though the parameters are tied across time, we can compute the partial derivatives with parameters at each time step assuming they can be independently changed. The actual gradient applied to the parameters is then the average of all of these gradients. Consequently, a simple way to visualize the gradient propagation through the network is to look at the gradient statistics over time steps, and seeing how they change as we train.

For the iRNN model trained using SGD with momentum, we show these visualizations for the training iterations at different intervals in Fig. 10. For each training iteration, we show the min, median and max of the gradients w.r.t the hidden to hidden connection matrix, for each time step of the RNN. We see that the gradient propagation is excellent at the beginning with almost steady decrease as we go backwards. This becomes substantially worse as we train. We see a quick drop in the backpropagated gradients and they remain zero thereafter. The situation improves near training iteration 225k and the gradient propagation looks better again. This is precisely the point where the iRNN starts to train well (Fig. 5). After this point, the gradients are more or less equally distributed. We observe that such gradient distribution is a good sign that the network has learned well.

We note that this visualization also reinforces our remark in Section 1 that even though the iRNN is initialized properly, subsequent learning takes it to a vanishing or exploding gradient regime, from which it takes a long time to recover.

In Fig. 11 we show similar visualizations for resRNN parameters W_{hh1} and W_{hh2} (Fig. 1). The gradients with W_{hh1} are identically zero at the first training iteration. This is due to the zero initialization of the network. The gradient distribution subsequently becomes close to the initial gradient plot of iRNN and maintains a similar shape throughout. The W_{hh2} distribution is balanced right from the beginning and stays the same as we train. The main takeaway from these visualizations is that residual networks are able to give gradient signals far back in time and hence train better than iRNNs.

7. Discussion and Future Work

VL-RNN training approach is strictly a superset of the RNN training. Indeed by making the penalty multiplier infinity, we exactly recover the RNN training algorithm. But it is unclear whether having a penalty multiplier equal to infinity right from the beginning (as with regular RNNs) is the right choice. The advantage of this approach is that it can potentially work on very long range sequences and also with a lot of activations including non saturating ones like ReLU. A disadvantage is that we have a separate matrix for each time step or set of time steps. This might need a lot of memory. Also, applying this model to real world data may bring other challenges not addressed here. In our experiments, we also found that the VL-iRNN training was extremely sensitive to the choice of penalty multiplier schedule, showing wild oscillations if we use too large a penalty too early. Figuring out a principled approach to use the penalty method is an interesting open problem.

It is unclear how resRNN should be compared to iRNN models. It may be relevant to compare them by using the same number of parameters. This however allows iRNN to have a larger hidden state. In our experiments, we chose to keep the size of the hidden state same for both models. We

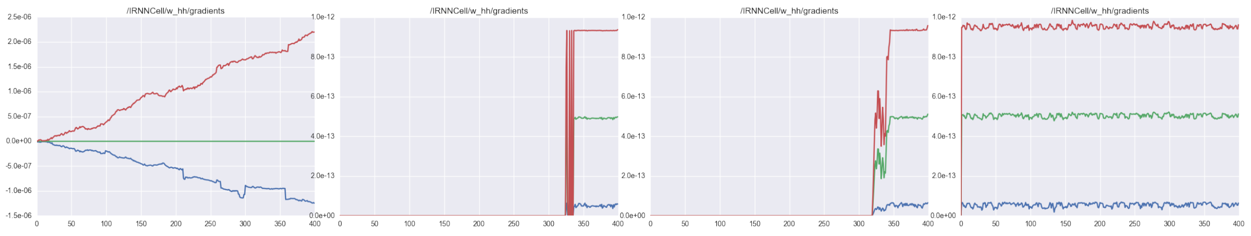
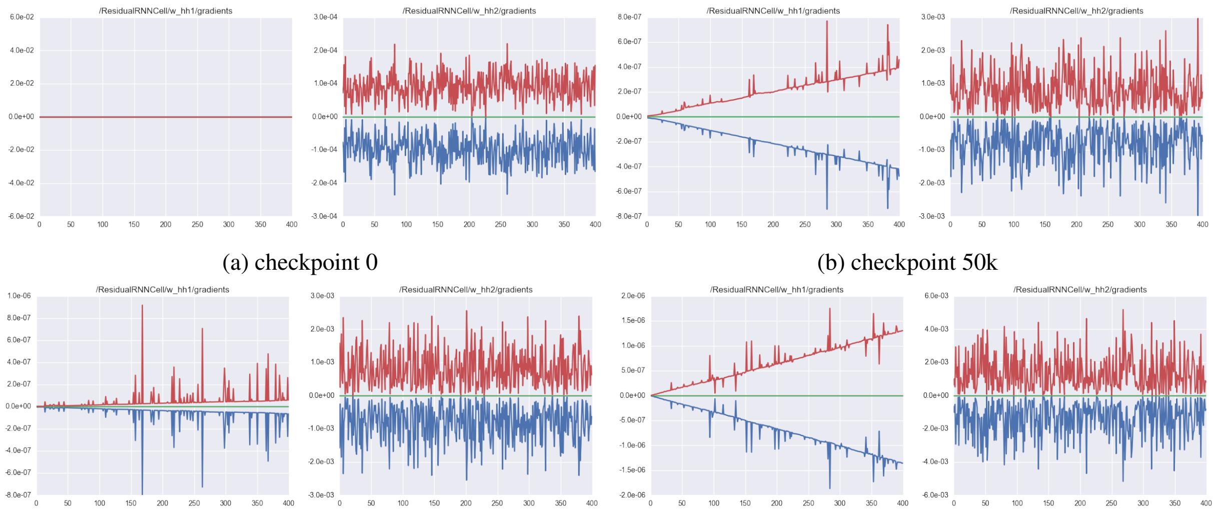

(a) checkpoint 0 (b) checkpoint 75k (c) checkpoint 150k (d) checkpoint 225k

Figure 10: iRNN gradient visualizations for W_{hh} : max (red), median (green), min (blue)


(a) checkpoint 0 (b) checkpoint 50k

(c) checkpoint 100k (d) checkpoint 150k

Figure 11: resRNN gradient visualizations for W_{hh} : max (red), median (green), min (blue)

leave the other experiment for future work.

We have not yet tried to test the limits of resRNN on the addition and multiplication tasks. It would be interesting to know the maximum sequence length for which residual recurrent networks can learn these tasks. Also, we did not train language models with larger number of hidden units due to memory constraints. Testing the limits of resRNN and iRNN for language modeling in terms of more hidden units would also be a direction of future work.

Document level language tasks such as summarization, classification and question answering rarely use the sequential nature of the data, primarily due to inability of the current recurrent models to learn very long term dependencies. Models that overcome this difficulty can potentially give improved performance on such tasks.

Getting and interpreting visualizations similar to the ones in the previous section, for VL-iRNNs and for other popular recurrent models is also left for future work.

8. Conclusion

In this work, we have presented a new training algorithm and a new recurrent architecture to learn long term dependencies. On two toy problems, we have shown improvements in the length of dependencies that can be learned as well as the speed of convergence. On a language modeling task, our models achieve a slightly better cross entropy loss. Finally, we have shown gradient visualizations which expose some of the inner workings of these algorithms.

References

[1] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[2] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[3] A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE in-*

ternational conference on acoustics, speech and signal processing, pages 6645–6649. IEEE, 2013.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[5] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

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

[7] M. Johnson, M. Schuster, Q. V. Le, M. Krikun, Y. Wu, Z. Chen, N. Thorat, F. Viégas, M. Wattenberg, G. Corrado, et al. Google's multilingual neural machine translation system: Enabling zero-shot translation. *arXiv preprint arXiv:1611.04558*, 2016.

[8] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311. ACM, 1984.

[9] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] D. Krueger and R. Memisevic. Regularizing rnns by stabilizing activations. *arXiv preprint arXiv:1511.08400*, 2015.

[11] Q. V. Le, N. Jaitly, and G. E. Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

[12] Q. Liao and T. Poggio. Bridging the gaps between residual learning, recurrent neural networks and visual cortex. *arXiv preprint arXiv:1604.03640*, 2016.

[13] L. Ma, Z. Lu, and H. Li. Learning to answer questions from image using convolutional neural network. *arXiv preprint arXiv:1506.00333*, 2015.

[14] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[15] T. Mikolov, I. Sutskever, A. Deoras, H.-S. Le, S. Kombrink, and J. Cernocky. Subword language modeling with neural networks. *preprint (http://www. fit. vutbr. cz/imikolov/rnnlm/char. pdf)*, 2012.

[16] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. *ICML (3)*, 28:1310–1318, 2013.

[17] S. Pradhan and S. Longpre. Exploring the depths of recurrent neural networks with stochastic residual learning.

[18] V. Ramanishka, A. Das, D. H. Park, S. Venugopalan, L. A. Hendricks, M. Rohrbach, and K. Saenko. Multimodal video description. In *Proceedings of the 2016 ACM on Multimedia Conference*, pages 1092–1096. ACM, 2016.

[19] I. V. Serban, R. Lowe, L. Charlin, and J. Pineau. Generative deep neural networks for dialogue: A short review. *arXiv preprint arXiv:1611.06216*, 2016.

[20] I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton. On the importance of initialization and momentum in deep learning. *ICML (3)*, 28:1139–1147, 2013.

[21] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[22] C. Szegedy, S. Ioffe, and V. Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261*, 2016.

[23] S. Venugopalan, H. Xu, J. Donahue, M. Rohrbach, R. Mooney, and K. Saenko. Translating videos to natural language using deep recurrent neural networks. *arXiv preprint arXiv:1412.4729*, 2014.

[24] Y. Wang and F. Tian. Recurrent residual learning for sequence classification.

[25] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*, 2(3):5, 2015.

[26] J. G. Zilly, R. K. Srivastava, J. Koutník, and J. Schmidhuber. Recurrent highway networks. *arXiv preprint arXiv:1607.03474*, 2016.