



DeepRSD: A Deep Regression Method for Sequential Data

Xishun Wang¹(✉), Minjie Zhang², and Fenghui Ren²

¹ Bitmain Technologies Inc., Beijing, China
xw357@uowmail.edu.au

² School of Computing and Information Technology, University of Wollongong,
Wollongong, Australia
{minjie, fren}@uow.edu.au

Abstract. Regressions on Sequential Data (RSD) are widely used in different disciplines. This paper proposes DeepRSD, which utilizes several different neural networks to result in an effective end-to-end learning method for RSD problems. There have been several variants of deep Recurrent Neural Networks (RNNs) in classification problems. The main functional part of DeepRSD is the stacked bi-directional RNNs, which is the most suitable deep RNN model for sequential data. We explore several conditions to ensure a plausible training of DeepRSD. More importantly, we propose an alternative dropout to improve its generalization. We apply DeepRSD to two different real-world problems and achieve state-of-the-art performances. Through comparisons with state-of-the-art methods, we conclude that DeepRSD can be a competitive method for RSD problems.

Keywords: Sequential data · Deep recurrent neural network
Alternative dropout · Regression

1 Introduction

In many regression problems, the predictor variables come from sequential data. For instance, predicting the hourly energy demand given weather forecasting in each hour, or estimating the valence of a facial expression from a video (sequence of images). Similar problems are widely seen in different disciplines. In this paper, we name these problems as Regressions on Sequential Data (RSD). RSD can be formally defined as follows.

$$\mathbf{y} = f_{\theta}(\mathbf{X}), \quad (1)$$

where f_{θ} is a transformation parameterized by θ , $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T]$ is a $T \times N$ matrix, representing T steps and N predictor variables (features) in each step, and \mathbf{y} can be a scalar or a T -dimension vector (a simple structured output).

X. Wang—Work done in University of Wollongong. Now the author is in Bitmain Technologies Inc.

The challenge of RSD is how to simultaneously model the (complex) features in each step and the temporal information. Gradient boosting [3] is an effective method to handle complex features, but it cannot naturally model temporal information. People have to make extra effort to encode temporal information when using gradient boosting. Continuous conditional random fields [16] can simultaneously model step features and temporal information, but it is relatively weak to deal with complex features. Feature engineering or other models are introduced [21] to compensate its weakness.

In this paper, we propose DeepRSD to solve the RSD problem. DeepRSD takes advantages of several variants of neural networks to construct an effective end-to-end learning method. In the family of neural networks, Recurrent Neural Networks (RNNs) [17] gain the capacity of modeling sequential data. There have been some variants of deep RNN [7, 14] in discrete problems. We deeply study different deep RNNs and choose stacked bi-directional RNN (Bi-RNN) to represent the sequential data \mathbf{X} efficiently. The stacked Bi-RNNs can learn not only the temporal transitions, but also the complex relations in the heterogeneous step feature \mathbf{x}_i through its nonlinear representations. As a result, stacked Bi-RNNs outperforms other deep RNNs in RSD problems.

It is challenging to train DeepRSD for two reasons. (1) RNN itself is hard to train. (2) Comparing to classification problems, the input and output are both unbounded in regression. We explore several conditions for a plausible training, including data preprocessing, initializations and preventing gradient vanishing/exploding for RNN. More importantly, we deeply study the dropout for stacked Bi-RNNs. The previous dropout methods do not work in DeepRSD. Instead, we propose an alternative dropout to effectively improve the generalization of DeepRSD. We also provide an explanation of alternative dropout through visualizations.

To show the advantages of DeepRSD, we insist to construct a universal model with an end-to-end learning manner to solve actual problems. We demonstrate the performance of DeepRSD in AMS solar energy prediction contest on Kaggle¹. To the best of our knowledge, DeepRSD gets the best result evaluated by Kaggle server. We further evaluate DeepRSD on a dataset of electricity demand prediction, from NPower Forecasting Challenge 2016². It is found that DeepRSD is still competitive even though the dataset is small. We also make general comparisons of DeepRSD and other state-of-the-art methods to demonstrate the effectiveness of DeepRSD.

The contributions of this paper are in three aspects: (1) We propose DeepRSD using several variants of neural networks for RSD problems and explore several conditions to reliably train DeepRSD. (2) We propose alternative dropout to effectively improve the generalization of DeepRSD. (3) We apply DeepRSD to two real-world problems and achieve state-of-the-art performances.

¹ <https://www.kaggle.com/c/ams-2014-solar-energy-prediction-contest>.

² <https://www.npowerjobs.com/graduates/forecasting-challenge>. Data are publicly available. Competition results are also published on this webpage.

2 Designs of DeepRSD

In this section, the standard RNN is briefly reviewed at first, and then the architecture of DeepRSD is illustrated. In the following, we discuss how to choose a proper activation function for DeepRSD. We also describe alternative dropout in detail.

2.1 RNN Review

As RNN is the core module for DeepRSD, we have a brief review of standard RNN [17]. For the input sequence $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T$, each in \mathbb{R}^N , RNN computes a sequence of hidden states $\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_T$, each in \mathbb{R}^M , and a sequence of predictions $\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, \dots, \hat{\mathbf{y}}_T$, each in \mathbb{R}^K , by iterating the equations

$$\mathbf{h}_i = \varphi_h(\mathbf{W}_{hx}\mathbf{x}_i + \mathbf{W}_{hh}\mathbf{h}_{i-1} + b_h) \quad (2)$$

$$\hat{\mathbf{y}}_i = \varphi_y(\mathbf{W}_{yh}\mathbf{h}_i + b_y) \quad (3)$$

where $\mathbf{W}_{hx}, \mathbf{W}_{hh}, \mathbf{W}_{yh}$ are weight matrices, b_h, b_y are bias terms, and φ_h, φ_y are activation functions. Equation 2 defines the input-to-hidden layer and Eq. 3 defines the hidden-to-output layer.

2.2 The Overall Architecture

Figure 1(a) illustrates the overall architecture of DeepRSD. We describe the network layers and their functions from bottom to top. DeepRSD consists of three modules: input processing module, main functional module and output processing module.

Input processing module includes the input layer and the Network-In-Network (NIN) layer [10]. The input layer is at the bottom, where sequential features are fed into DeepRSD. On the top of the input layer is the NIN layer that reduces the dimension of features using a linear activation function. The NIN layer contributes to accelerating the training process; meanwhile, NIN layer does not affect the precision of final predictions.

Main functional module is a stack of Bi-RNN layers with different sizes. RNN is suitable to model sequential data because it introduces hidden layer to encode the temporal information. Bi-RNN [8] can effectively represent the correlations, and has been widely used to model sequential data. The structure of Bi-RNN is illustrated in Fig. 1(b). The stacked Bi-RNNs supply sufficient nonlinearities for the sequence and also for step feature \mathbf{x}_i . Even the features in \mathbf{x}_i are heterogeneous and have complex functional relationships, the stacked Bi-RNNs can learn to represent them automatically.

Output processing module is a step-wise dense layer that outputs the predicted value at each step. In Fig. 1(a), we show the unrolled T dense layers, corresponding to the T predictions. The T predictions can also be added up for a scalar prediction if necessary. Therefore, the final output \mathbf{y} can be a T -dimension vector or a simple scalar.

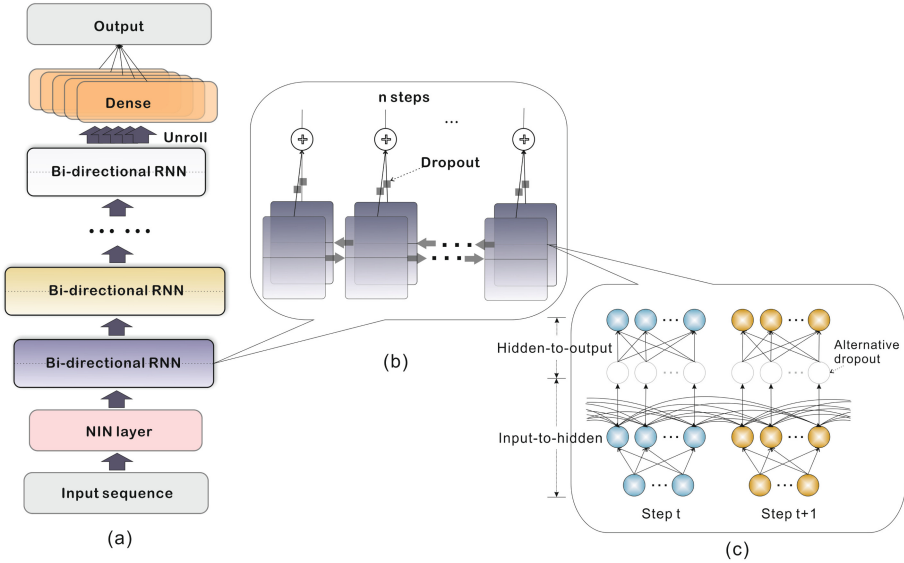


Fig. 1. The overall architecture of DeepRSD

Stacked Bi-RNNs, the main functional module in DeepRSD, is designed catering the characteristics of RSD problems. In discrete problems, Pascanu et al. [14] discussed how to construct deep RNN and proposed three variants of deep RNN, which are deep input-to-hidden, deep hidden-to-output and deep transition networks. Graves [6] used RNN with stacked hidden layers to generate sequences. Much effort in RNN is made to handle long dependency in sequences in discrete problems, while in RSD, we aim to model the sequential information as well as the functional relationships among step features. For many RSD problems in practice, the sequential data is not so long that a standard hidden layer is adequate to handle the dependencies, thus no necessary to introduce deep transitions. In the stacked Bi-RNNs, there are dense connections in input-to-hidden and hidden-to-output layers, which supply plenty of nonlinearities to encode the heterogeneous step features. We tried deep input-to-hidden and deep hidden-to-output networks, and neither of them worked as well as stacked Bi-RNNs in our experimented datasets. The advantages of stacked Bi-RNNs over other deep RNNs are further shown in experiments in Subsect. 4.1.

2.3 Activation Function

For the nonlinear activation function, we use leaky rectified linear unit (leaky ReLU) [1], shown in Eq. 4.

$$\varphi(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases} \quad (4)$$

For RNN, traditional activations—sigmoid or hyperbolic tangent functions, which have saturated zones, may lead to the gradient vanishing problem [15]. ReLU [12] is advantageous since (1) it alleviates the vanishing gradient and (2) it offers a simple gradient computation. However, it often makes DeepRSD diverge. When using leaky ReLU, we can train DeepRSD more reliably. For the above reasons, we choose leaky ReLU as the activation function for DeepRSD.

2.4 Alternative Dropout

Dropout [19] has been a simple and effective way to improve the generalization of feed-forward neural networks. However, people struggle to develop effective dropout for RNN. Zaremba et al. [22] applied dropout to the input and hidden-to-output but not to the transition layer in RNN. Gal et al. [4] proposed a theoretically grounded dropout to the hidden layer in RNN and got improved performance. These methods work effectively for discrete problems. However, they do not work in DeepRSD.

Considering the structure of stacked Bi-RNNs, we propose a special dropout scheme. For the standard RNN shown in Fig. 1(c), we apply dropout between the input-to-hidden and hidden-to-output layers, while for the stacked second Bi-RNN, this dropout is applied alternatively. To be specific, dropout is applied only to the *forward* RNN for the first Bi-RNN. For the stacked second Bi-RNN, dropout is applied to the *backward* RNN. Dropout is applied alternatively in the forward and backward directional RNNs, so we name this dropout scheme as alternative dropout.

For the stacked Bi-RNNs, the alternative dropout between input-to-hidden and hidden-to-output layer is crucial to improve generalization. We supply a possible explanation for alternative dropout. We assume that the input-to-hidden layer of forward RNN outputs a vector in range $[a, b]$. The hidden-to-output layer is simplified as an identity map. If we do not apply any dropout, Bi-RNN sums the outputs of each direction and thus gives an output range of approximate $[2a, 2b]$ (the output range of two RNNs should be similar). For dropout with a keep rate γ ($\gamma < 1$), to keep the invariance of expectation, we re-scale the vector and lead to an inflated approximate range $[a/\gamma, b/\gamma]$ (Re-scaling is the default manner in dropout [19]). If dropout with a keep rate γ is applied to both directions, the output range of Bi-RNN will be $[2a/\gamma, 2b/\gamma]$. In contrast, for alternative dropout, the output range of Bi-RNN will be $[a/\gamma + a, b/\gamma + b]$. We can see that the dropout on both directions leads to a larger fluctuation range than alternative dropout. The large fluctuations may bring difficulty to optimization algorithms to find a good local minimum. We experiment and visualize alternative dropout in Subsect. 4.1 to support our explanation.

3 Training and Inference

We first set the cost function for DeepRSD. For regression problems, Mean Square Error (MSE) and Mean Absolute Error (MAE) are widely used. As MAE

is more robust to very large or small values, we use MAE as the cost function for DeepRSD. If the output \mathbf{y} is a T -dimension vector, the cost function can be written as:

$$C(\theta) = \frac{1}{ST} \sum_i^S \sum_i^T |y_i - \hat{y}_i|, \quad (5)$$

where S is the total number of sequence samples, and θ is parameters to be learned (see Eq. 1). If \mathbf{y} is a scalar, the summation \sum_i^T can be omitted.

DeepRSD is hard to train for several reasons. Comparing to classification problems, the inputs and outputs are both unbounded in regression problems, which may lead to divergence in the training of DeepRSD. Moreover, improper initializations will result in divergence or invalid learning in deep networks. Besides, there are gradient vanishing/exploding problems in (deep) RNN, which may hamper the training process. In our study, we find the conditions in Table 1 are necessary to ensure a plausible training.

Table 1. Conditions for DeepRSD training

Condition	Description
Data preprocessing	Normalize and trim data
Initializations	Introduce proper initializations for RNN
Gradient vanishing preventing	Use leaky ReLU activation
Gradient exploding preventing	Use gradient clipping

Data Preprocessing. Sutskever et al. [20] pointed out it is essential to normalize data (both inputs and outputs) to obtain a reliable training for RNN. We follow this rule and normalize the inputs and outputs for DeepRSD. We further trim the extreme values to result in a bounded region $[-a, a]$. If we set $a = 5$, the extreme values outside the 5σ region of Gaussian will be trimmed.

Initializations. Initializations are critical to ensure the convergence of RNN [20]. We use Gaussian and orthogonal initializations [5, 18] for the stacked Bi-RNNs. A correct scale of Gaussian is essential for a plausible training. For normalized data, we find $\sigma = 0.01$ is a good scale to ensure convergence.

Gradient Vanishing Preventing. RNN is difficult to train because they suffer from gradient vanishing/exploding problems [15]. We use leaky ReLU activation to prevent vanishing gradients(see Subsect. 2.3).

Gradient Exploding Preventing. We simply use gradient clipping [15] to prevent gradient exploding for RNN.

Applying the above conditions, DeepRSD can be trained reliably. We resort to mini-batch Stochastic Gradient Decent (SGD) with Nesterov’s momentum [13], with learning rate decay [20], to train DeepRSD. We also try other optimization algorithm, such as Adadelta [23] and Adam [9], but the results are not so good as SGD with Nesterov’s momentum. Therefore, we recommend to use SGD with Nesterov’s momentum to train DeepRSD.

Inference in DeepRSD can be quite efficient. As DeepRSD introduces dropout, we simply employ the inference process of dropout networks to predict \mathbf{y} [19].

4 Experiment

DeepRSD is evaluated on two different real-world problems from data science competitions. We apply DeepRSD to the AMS solar energy prediction problem to predict a scalar value. We further evaluate DeepRSD on an electricity demand prediction problem, where the output is structured.

For both problems, we used DeepRSD to construct a universal end-to-end learning model. We used the same conditions in training for the two problems. In data preprocessing, the data were normalized to standard Gaussian. We set the bound for trimming, $a = 5$. To cope with the gradient vanishing problem, DeepRSD used leaky ReLU activation with a leakiness $\alpha = 0.15$. For gradient exploding, DeepRSD used gradient clipping to normalize the gradients if their norm exceeded 1.0 [15]. In initializations, for the transition matrix W_{hh} in the RNN layer, DeepRSD used orthogonal initialization [18] with a gain $(1 + \alpha^2)^{\frac{1}{2}}$. For the input-to-hidden weight matrix W_{hx} , DeepRSD used Gaussian distribution with 0 mean and 0.01 standard variance to initialize it.

4.1 Evaluations on AMS Solar Energy Prediction Contest

The objective of the problem is to predict the daily incoming solar energy at 98 Oklahoma Mesonet sites [11]. Detail information can be found on Kaggle’s website (see footnote 1).

Solutions Using DeepRSD. We use DeepRSD to construct one universal model for the 98 Mesonet sites. For each Mesonet site, we extract the weather data from its neighbouring four GEFS grids. We also supply the distances of the Mesonet site to its neighbouring four grids. Besides, we add the temporal and the Mesonet site specific information. All the above information is combined to be a step feature with 77 variables. The raw features are input to DeepRSD, without any feature engineering. DeepRSD uses alternative dropout to improve its generalization. DeepRSD not only learns the transitions in weather forecasting sequences, but also hiddenly learns to approximate the weather states at the Mesonet site from the heterogeneous features.

The hyperparameters were set using random search [2], followed by handcraft tuning. The input dimension was 77 for the input layer. The output dimension of NIN layer was 64, reducing the input dimension by 16.9%. DeepRSD introduced 4 stacked Bi-RNN and dense layers, with the size 128, 224, 128 and 16. The following dense layer had an output dimension of 1. The last layer was a mean-pooling layer to output the daily production. The dropout rate for both forward and backward RNN before the summation was 0.3, and the drop rate of alternative dropout was 0.5. Dropout was applied in the first 3 layers of the stacked Bi-RNNs.

Test Results. Table 2 shows DeepRSD’s result measured by MAE and the top three results in the Kaggle competition. We can see that GBDT is the dominant method that holds the top three places in the past competition. The top three methods used complex feature engineering, post-processing or model combinations to improve the results [11]. In contrast, DeepRSD simply averaged 10 differently trained models and got an MAE of 2.10M.

Table 2. Ours and the top three results in the Kaggle contest

Solution	MAE (MW)	Method
1 st Place	2.11	GBDT
2 nd Place	2.13	GBDT
3 rd Place	2.16	GBDT
<i>Ours</i>	<i>2.10</i>	<i>DeepRSD</i>

Comparing DeepRSD to Other Deep RNNs. In this experiment, we compare the performance of deep RNNs. Besides stacked Bi-RNNs, we also constructed deep input-to-hidden (DeepIH) network and deep hidden-to-output (DeepHO) network. We used DeepIH and DeepHO to replace stack Bi-RNNs in the architecture of DeepRSD. The other parts of DeepRSD stayed the same. Besides, we also use standard RNN and multi-layer dense neural networks (DNN) as the baseline.

The configurations of DeepRSD stayed the same as introduced in the second paragraph in Sect. 4. The configurations of DeepIH and DeepHO were optimized by random search [2]. DeepIH used two dense layers, with size 32 and 32, for each time step; and two RNN layers, with size 192 and 128. DeepHO used two RNN layers with size 128 and 128, followed by two dense layers with size 128 and 64. The DNN had 4 dense layers, with size 128, 256, 192 and 128. The standard RNN had a size of 128. The performances of the five different RNNs are shown in Fig. 2.

In Fig. 2, we use standard RNN and DNN as the baseline. RNN obtains an MAE of 2.20 MW, and DNN obtains an MAE of 2.21 MW. Though DNN is a deep network that has 4 layers, it does not utilize the temporal information, and therefore has a low performance. Comparing to the two baselines, DeepIH, DeepHO and DeepRSD make significant improvements on this problem.

We then analyze the three deep RNNs in detail. For DeepIH, it first uses dense networks to represent step features, and then input the representations into RNN. For DeepHO, it first represents features using two layers of RNNs, and then uses dense network to refine the output of RNN. In contrast, DeepRSD, which uses stacked Bi-RNNs, always combines the nonlinear representations of step features and temporal information. We can see that DeepRSD has advantages over DeepIH and DeepHO in Fig. 2. That is the reason we choose stacked Bi-RNNs for sequential data regression.

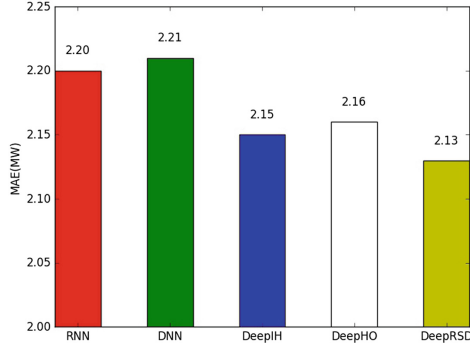


Fig. 2. The performance comparisons of different networks

Analysis of Alternative Dropout. In this experiment, we compare and analyze three dropout ways for DeepRSD. The first way is naive dropout, where dropout is only applied in the input projection and output projection [22]. The second way is variational dropout, where dropout is applied to the hidden layer besides the input and output projections [4]. The third way is the proposed alternative dropout. The three dropout methods are applied to DeepRSD with the same other configurations.

The MAEs and epochs to converge are shown in Table 3 for the three dropout methods. Without any dropout, DeepRSD achieves an MAE of 2.17 MW. Though different dropout rates have been tried, naive dropout and variational dropout do not improve the generalization of DeepRSD. Moreover, variational dropout converges very slow, up to 40 epochs. Alternative dropout effectively enhances the generalization of DeepRSD by near 2%. In discrete problems, networks with dropout will consume more training time. This is also seen in regression problems in Table 3. DeepRSD without dropout converges in 4 epochs, while DeepRSD with alternative dropout converges in 6 epochs.

Table 3. Comparisons of different dropout ways

Methods	MAE (MW)	Epoches to converge
No dropout	2.17	4
Naive dropout	2.17	6
Variational dropout	2.17	40
Alternative dropout	2.13	6

We intuitively explain the alternative dropout via data visualizations. When the network was near convergence in training, we collected the output of the first step of the third Bi-RNN. The output was sampled every 100 mini-batches from batch 30000 to 35000. In the same manner, we visualized the output using naive

dropout. The visualization of alternative dropout and naive dropout are shown in Fig. 3a and b, respectively. We can see that the outputs are consistent for the alternative dropout in Fig. 3a, while the outputs fluctuate in the naive dropout in Fig. 3b. The inconsistency and fluctuations in traditional dropout bring difficulty for the optimization algorithms, resulting in a poor local minimum. Even though we decrease the dropout rate, naive dropout does not work well. In contrast, alternative dropout provides a mild dropout way for stacked Bi-RNNs.

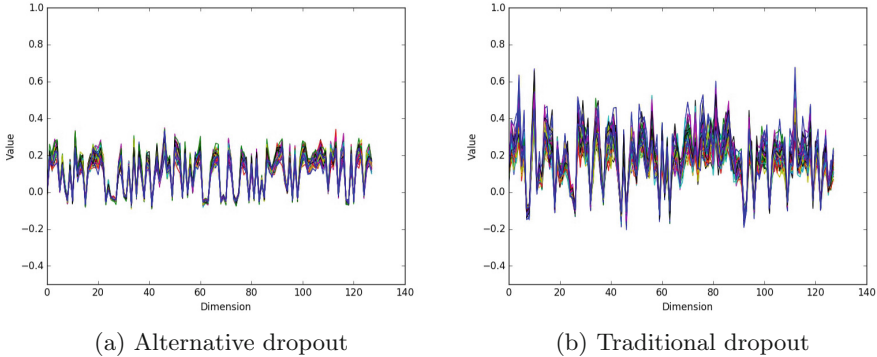


Fig. 3. Data visualization of dropout

4.2 Evaluations on Electricity Demand Forecasting Competition

The task of this competition is to predict the future power demand in every half hour according to weather data. Detail information can be found on NPower Demand Forecasting Challenge website (see footnote 2).

Solutions Using DeepRSD. We use DeepRSD to model this sequential data prediction problem. The extracted raw features include weather feature, temporal feature, and calendar feature, 13 variables in total. We use a data sequence with length 48, which corresponds to electricity usages of half hour in a day. This competition used Mean Absolute Percentage Error (MAPE) for the final score, while we can still use MAE as the cost function to train DeepRSD.

In this problem, the hyperparameters for DeepRSD were set as follows. The input dimension was 13. DeepRSD introduced 2 stacked Bi-RNNs with the size 16 and 16. The following dense layer had an output dimension of 1. The final output was a 48-dimension vector. DeepRSD used alternative dropout with a drop rate of 0.2.

Results and Comparisons. We compare the performance of DeepRSD with the winning methods and several state-of-the-art methods in this competition. The results, measured by MAPE, are shown in Table 4.

Table 4. Evaluation results of the two rounds in NPower Forecasting Challenge 2016.

Model	MAPE in round 1	MAPE in round 2
1 st Place	3.14%	7.13%
2 nd Place	6.43%	4.89%
3 rd Place	7.84%	7.48%
ARIMA	8.95%	8.77%
SGCRF	5.83%	6.64%
GBDT	5.40%	4.97%
DeepRSD	(4.88 ± 0.21)%	(4.56 ± 0.20)%

The evaluations are based on a rolling forecasting mode, same as the competition. The top three methods did not employ complex machine learning methods, but relied heavily on feature engineering³. ARIMA is employed as the baseline model, which is not as competitive as the top three methods.

The following methods are popular in recent. The overall performance of SGCRF is comparable to the second place in the competition. The results of GBDT are close to the first place. DeepRSD achieves the best performance, with an average MAPE of 4.72%, better than the first place. Besides, DeepRSD only uses the raw feature without any feature engineering. The results of DeepRSD fluctuates due to the instability of neural networks, but this can be compensated by model ensemble.

5 Conclusion

In this paper, we proposed DeepRSD for the regressions on sequential data. DeepRSD used stacked Bi-RNNs to represent the sequential data. We pointed out four conditions to ensure a plausible training. We also proposed an alternative dropout to effectively improve the generalization of DeepRSD. We applied DeepRSD to two real-world sequential data prediction problems and achieved state-of-the-art performances. According to the experimental results, DeepRSD showed two major advantages over other methods. (1) DeepRSD can simultaneously represent step features and temporal information. (2) DeepRSD has strong nonlinear presentation capacity to achieve a good performance without feature engineering. Therefore, we conclude that DeepRSD can be an effective solution for regressions on sequential data.

³ <http://blog.drhongtao.com/2016/12/winning-methods-from-npower-forecasting-challenge-2016.html>.

References

1. Agostinelli, F., Hoffman, M., Sadowski, P., Baldi, P.: Learning activation functions to improve deep neural networks. arXiv preprint [arXiv:1412.6830](#) (2014)
2. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. *J. Mach. Learn. Res.* **13**(Feb), 281–305 (2012)
3. Chen, T., Guestrin, C.: XGBoost: a scalable tree boosting system. arXiv preprint [arXiv:1603.02754](#) (2016)
4. Gal, Y.: A theoretically grounded application of dropout in recurrent neural networks. arXiv preprint [arXiv:1512.05287](#) (2015)
5. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: AISTATS, vol. 9, pp. 249–256 (2010)
6. Graves, A.: Generating sequences with recurrent neural networks. arXiv preprint [arXiv:1308.0850](#) (2013)
7. Graves, A., Jaitly, N., Mohamed, A.-R.: Hybrid speech recognition with deep bidirectional LSTM. In: 2013 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU), pp. 273–278. IEEE (2013)
8. Graves, A., Schmidhuber, J.: Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Netw.* **18**(5), 602–610 (2005)
9. Kingma, D., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint [arXiv:1412.6980](#) (2014)
10. Lin, M., Chen, Q., Yan, S.: Network in network. arXiv preprint [arXiv:1312.4400](#) (2013)
11. McGovern, A., Gagne, D.J., Basara, J., Hamill, T.M., Margolin, D.: Solar energy prediction: an international contest to initiate interdisciplinary research on compelling meteorological problems. *Bull. Am. Meteorol. Soc.* **96**(8), 1388–1395 (2015)
12. Nair, V., Hinton, G.E.: Rectified linear units improve restricted Boltzmann machines. In: Proceedings of the 27th International Conference on Machine Learning (ICML 2010), pp. 807–814 (2010)
13. Nesterov, Y.: A method of solving a convex programming problem with convergence rate $\mathcal{O}(1/k^2)$. In: Soviet Mathematics Doklady, vol. 27, pp. 372–376 (1983)
14. Pascanu, R., Gulcehre, C., Cho, K., Bengio, Y.: How to construct deep recurrent neural networks. arXiv preprint [arXiv:1312.6026](#) (2013)
15. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks. In: ICML, vol. 28, no. 3, pp. 1310–1318 (2013)
16. Qin, T., Liu, T.-Y., Zhang, X.-D., Wang, D.-S., Li, H.: Global ranking using continuous conditional random fields. In: Advances in Neural Information Processing Systems, pp. 1281–1288 (2009)
17. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. *Cognit. Model.* **5**(3), 1 (1988)
18. Saxe, A.M., McClelland, J.L., Ganguli, S.: Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. arXiv preprint [arXiv:1312.6120](#) (2013)
19. Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **15**(1), 1929–1958 (2014)
20. Sutskever, I., Martens, J., Dahl, G.E., Hinton, G.E.: On the importance of initialization and momentum in deep learning. In: ICML, vol. 28, no. 3, pp. 1139–1147 (2013)

21. Wytock, M., Kolter, Z.: Sparse Gaussian conditional random fields: algorithms, theory, and application to energy forecasting. In: International Conference on Machine Learning, pp. 1265–1273 (2013)
22. Zaremba, W., Sutskever, I., Vinyals, O.: Recurrent neural network regularization. arXiv preprint [arXiv:1409.2329](https://arxiv.org/abs/1409.2329) (2014)
23. Zeiler, M.D.: ADADELTA: an adaptive learning rate method. arXiv preprint [arXiv:1212.5701](https://arxiv.org/abs/1212.5701) (2012)