The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Sparse Gaussian Conditional Random Fields on Top of Recurrent Neural Networks

Xishun Wang, Minjie Zhang, Fenghui Ren
School of Computing and Information Technology,
University of Wollongong, 2522, NSW, Australia.
xw357@uowmail.edu.au; {minjie, fren}@uow.edu.au

Abstract

Predictions of time-series are widely used in different disciplines. We propose CoR, Sparse Gaussian **C**onditional Random Fields (SGCRF) **o**n top of **R**ecurrent Neural Networks (RNN), for problems of this kind. CoR gains advantages from both RNN and SGCRF. It can not only effectively represent the temporal correlations in observed data, but can also learn the structured information of the output. CoR is challenging to train because it is a hybrid of deep neural networks and densely-connected graphical models. Alternative training can be a tractable way to train CoR, and furthermore, an end-to-end training method is proposed to train CoR more efficiently. CoR is evaluated by both synthetic data and real-world data, and it shows a significant improvement in performance over state-of-the-art methods.

Introduction

Predictions of time-series data occur widely in financial analysis, market demand prediction and video analysis (Box et al. 2015). This paper studies multi-step prediction of time-series data, which can be formally defined as follows:

$$\mathbf{y} = f_W(\mathbf{X}), \tag{1}$$

where f_W is a transformation (to be learned) parameterized by W, $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_T]$ is a $T \times D$ matrix, representing T steps and D observed features in each step, and $\mathbf{y}$ is a T-dimension structured output.

The challenge of this problem comes from two aspects: 1) There are temporal and nonlinear correlations in the observed data $\mathbf{X}$; 2) The predicted sequence $\mathbf{y}$ is inherently structured. AutoRegressive Integrated Moving Average (ARIMA) is a traditional model for time-series prediction (Box and Pierce 1970), which utilizes the temporal correlations of the observed data and output for prediction. Gradient boosting (Friedman 2001) has shown competitive performance for regression problems. However, gradient boosting calls for extra effort (usually by feature engineering) to model time-series data. Recently, deep Recurrent Neural Networks (RNN) (Pascanu et al. 2013) and Sparse Gaussian Conditional Random Fields (SGCRF) (Wytock and Kolter 2013) have been introduced to time-series prediction, showing promising results.

SGCRF is an effective structured learning model, while it is weak in representing nonlinearities in features. As a consequence, SGCRF relies heavily on feature engineering. RNN can effectively model time-series data and represent heterogeneous features, while it misses the structured information in the output. This paper proposes Sparse Gaussian **C**onditional Random Fields **o**n top of **R**ecurrent Neural Networks (CoR) for multi-step time-series prediction. CoR shows several advantages over RNN and SGCRF. Compared to RNN, CoR can learn structured information and thus lead to a significant boost in performance. Compared to SGCRF, CoR can effectively represent heterogeneous temporal features and greatly outperform SGCRF. Moreover, CoR consumes many fewer computational resources than SGCRF.

CoR has a deep architecture as illustrated in Figure 1. To enhance the representational capacity, we build stacked Bi-directional RNNs. On top of the deep RNNs is then SGCRF. The challenges of training CoR come from two aspects: 1) CoR is a complex hybrid of deep neural networks and densely-connected graphical models; 2) The training of CoR is a constrained optimization problem, where the precision matrix in SGCRF must be positive-definite. It is natural to resort to alternative training for CoR, in which RNN and SGCRF are trained alternatively until they converge. Furthermore, we propose an end-to-end training method for CoR based on projected gradient descent (Lin 2007). Compared to alternative training, end-to-end training is more efficient and results in better prediction accuracy.

In experiments, the advantages of CoR over stacked Bi-RNNs and SGCRF are demonstrated via synthetic data. Under varying conditions of synthetic data, we also discover several interesting properties of SGCRF, stacked Bi-RNNs and CoR. CoR is also evaluated by real-world data from an electricity demand prediction competition, and it shows better performance than other state-of-the-art methods.

The contributions of this paper may be summarized in three aspects. 1) We propose a new model called CoR for multi-step time-series prediction. CoR gains advantages from RNN and SGCRF. It can simultaneously represent nonlinear temporal features and capture the structured information of the output. 2) CoR is a complex hybrid model that is challenging to train. We propose two methods, alternative training and end-to-end training, to train CoR. In comparison, the end-to-end training is more effective. 3) CoR shows a significant improvement in performance over other models in multi-step time-series prediction. The performance boost of CoR suggests that both temporal correlations and structured output information are important in multi-step time-series prediction.

Related Work

Prediction of time-series data is an important topic and has been studied for decades. ARIMA (Box and Pierce 1970) is a traditional algorithm and has been successfully applied in different disciplines, such as financial analysis and demand prediction (Box et al. 2015). ARIMA makes use of the temporal correlations in observed data and output, yet it is not sophisticated enough compared to recent models and thus is less competitive in performance.

Gradient boosting (Friedman 2001) is a strong ensemble model and recently shows competitive performance in regression problems (Chen and Guestrin 2016). However, gradient boosting does not naturally model time-series data. Extra feature engineering has to be conducted when applying gradient boosting to time-series prediction.

RNN (Pineda 1987) can effectively model sequential data and therefore may be used in time-series predictions. Deep RNN architectures can enhance representational capacity and show competitive performance in time-series prediction (Lin, Guo, and Aberer 2017). Though deep RNNs are powerful, they miss the structured output information. Therefore, there is still room to improve deep RNNs.

SGCRF (Wytock and Kolter 2013) improves fully-connected graphical models by pruning redundant connections. It can effectively learn salient-structured information and has achieved success in predictions of time-series data. However, SGCRF is a log-linear model that relies heavily on feature engineering. Moreover, SGCRF consumes a very large computation resource for large problems. McCarter and Kim (McCarter and Kim 2016) made efforts to reduce computational cost on large-scale learning of SGCRF.

Different from previous models, we propose CoR, an integration of RNN and SGCRF, for time-series prediction. CoR combines advantages from RNN and SGCRF, so it can simultaneously represent temporal features and learn structured information. We demonstrate that CoR significantly outperforms the aforementioned models in real-world data predictions.

Deep neural networks have achieved great success in various fields, such as computer vision and natural language processing (Bengio and others 2009). However, neural networks cannot directly model structured information. Recently, there have been efforts to combine deep neural networks with graphical models. Zheng et al. (Zheng et al. 2015) combined convolutional neural networks with conditional random fields for semantic segmentation and achieved improved performance. Lample et al. (Lample et al. 2016) integrated Long Short-Term Memory (LSTM) and conditional random fields for named entity recognition. Our work integrates RNN and SGCRF for time-series prediction. Compared to previous research, we are developing a different model to solve the problem in another field.

There are some very recent studies of time-series prediction based on deep neural networks. Osogami and Otsuka (Osogami and Otsuka 2015) proposed Dynamic Blotzmann Machine (DyBM) for time-series prediction. The most notable character of DyBM is online updating, while its performance is not as good as RNN in off-line learning. Lin et al. (Lin, Guo, and Aberer 2017) used attention-based RNN and introduced dual-stage attention to improve the accuracy of time-series prediction. Their work tried to model the observed data in a more effective way, but lacked consideration of structured information of output.

Architecture of CoR

The overall architecture of CoR is illustrated in Figure 1. CoR has two modules: the bottom part is stacked bi-directional RNNs (Bi-RNNs); and the top part is SGCRF. $\mathbf{X} \in \mathbb{R}^{T\times D}$ is the input observation. $\mathbf{y} \in \mathbb{R}^{T}$ is the final output, which corresponds to the observation in each time step. $\mathbf{z} \in \mathbb{R}^{T}$ is an intermediate variable, which is the output of stacked Bi-RNNs and the input of SGCRF. In the following sections, the stacked Bi-RNNs and SGCRF are described in detail.

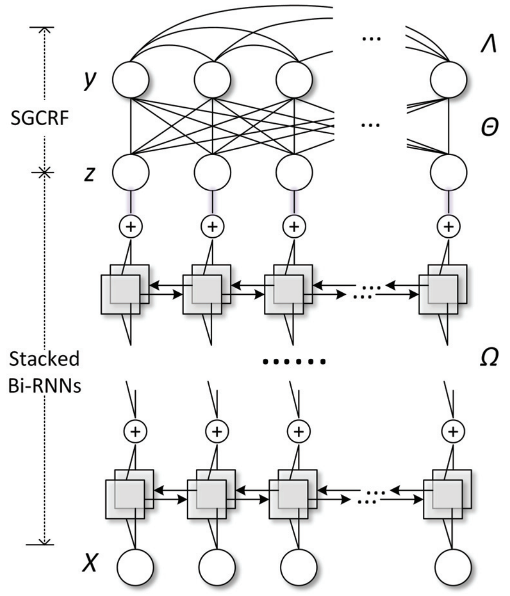

Figure 1: The overall architecture of CoR. The bottom part is stacked Bi-RNNs, and the top part is SGCRF. **X** is the input observation, **y** is the final output variable, and **z** is an intermediate variable, which is the output of stacked Bi-RNN and the input of SGCRF. Ω denotes all the parameters of stacked Bi-RNNs, while Θ and Λ are parameters of SGCRF.

Stacked Bi-RNNs

To illustrate the stacked Bi-RNNs, we start from a standard RNN (Pineda 1987). An input sample $\mathbf{X} \in \mathbb{R}^{T\times D}$ is a data sequence. For the sequence $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_T$, each in $\mathbb{R}^D$, RNN computes a sequence of hidden states $\mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_T$, each in $\mathbb{R}^M$, and a sequence of predictions $\hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2, \cdots, \hat{\mathbf{z}}_T$, each in $\mathbb{R}^K$, by iterating the equations

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

$$\hat{\mathbf{z}}_t = \varphi_z(\mathbf{W}_{zh}\mathbf{h}_t + b_z) \quad (3)$$

where $\mathbf{W}_{hx}, \mathbf{W}_{hh}, \mathbf{W}_{zh}$ are weight matrices, b_h, b_z are bias terms, and φ_h, φ_z are activation functions.

Bi-RNN is a combination of two standard RNNs in reversed directions. The performance of Bi-RNN has been better and more stable than a standard RNN in speech recognition (Graves and Schmidhuber 2005). To adapt Bi-RNN for regression, the ouputs of two reversed RNNs are added together as the output of Bi-RNN. Multiple Bi-RNNs can be stacked to construct a deep network to further enhance representational capacity. We build stacked Bi-RNNs networks, shown in Figure 1, as the bottom part of CoR. As the output of stacked Bi-RNN is in $\mathbb{R}^K$ in each time step, a dense layer is added to reduce $\mathbb{R}^K$ to $\mathbb{R}^1$ for the last Bi-RNN. The output $\mathbf{z}$ of stacked Bi-RNNs can be formulated as

$$\mathbf{z} = f_\Omega(\mathbf{X}), \tag{4}$$

where f_Ω is the learned mapping parameterized by Ω.

The architecture of stacked Bi-RNNs can be flexible. If the amount of training data is small, it can be reduced to a single RNN. If the number of time steps is large, Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber 1997) can be introduced to replace standard RNN, while the other structures remain. The flexibility of stacked Bi-RNNs enables CoR to apply to different real-world time-series prediction problems.

SGCRF

Gaussian CRF is a graphical model that can effectively model the structured information of output. It takes $\mathbf{z}$ as input and ouputs $\mathbf{y}$. We resort to the energy model to illustrate Gaussian CRF. The energy function $E(\mathbf{z}, \mathbf{y})$ has two terms,

$$E(\mathbf{z}, \mathbf{y}) = 2\mathbf{z}^T\Theta\mathbf{y} + \mathbf{y}^T\Lambda\mathbf{y}, \tag{5}$$

where the first term maps $\mathbf{z}$ to $\mathbf{y}$, parameterized by Θ, and the second term models the conditional dependencies of $\mathbf{y}$, parameterized by Λ. The inverse covariance matrix Λ is constrained to be positive-definite to ensure a valid multivariate Gaussian. The resultant Gaussian CRF is formulated as

$$p(\mathbf{y}|\mathbf{z}) = \frac{1}{Q(\mathbf{z})} exp\{-E(\mathbf{z}, \mathbf{y})\}, \tag{6}$$

where $Q(\mathbf{z})$ is the partition function,

$$\frac{1}{Q(\mathbf{z})} = c|\Lambda| exp\{-\mathbf{z}^T\Theta\Lambda^{-1}\Theta^T\mathbf{z}\}. \tag{7}$$

Parameter Θ encodes dense dependencies of $\mathbf{y}$ on $\mathbf{z}$, and parameter Λ also introduces dense conditional dependencies on $\mathbf{y}$. These dense dependencies are illustrated in Figure 1. It was observed that the connections are redundant and may cause overfitting (Wytock and Kolter 2013). Therefore, constraints for sparsity are applied to these dependencies by using L_1 norm to Θ and Λ in the training process. Gaussian CRF with sparse dependencies is thus called SGCRF.

The stacked Bi-RNNs offer powerful nonlinearities that can effectively represent the temporal and heterogeneous features, while SGCRF can model the structured output information. CoR, an integration of stacked Bi-RNNs and SGCRF, can take advantage of the two models for multi-step time-series prediction.

Training CoR

We propose two training methods for CoR, alternative training and end-to-end training. The following denotations are used in both training methods. Assuming there are N training samples, $\mathcal{X} = \{\mathbf{X}^{(1)}, \cdots, \mathbf{X}^{(N)}\}$ denotes the whole training feature set. $Y = \{\mathbf{y}^{(1)}, \cdots, \mathbf{y}^{(N)}\}$ denotes the N predictions, while $\dot{Y}$ denotes the corresponding N ground-truths. Z denotes the N intermediate predictions by stacked Bi-RNNs, which is also the input of SGCRF.

Alternative training

As CoR has two modules, it is natural to train the two modules alternatively. Alternative training first trains the stacked Bi-RNNs with Mean Absolute Error (MAE), and then trains SGCRF by minimizing its negative log-likelihood. Finally, it trains stacked Bi-RNNs and SGCRF alternatively until convergence, using a common loss function.

1. Initial training of stacked Bi-RNNs We first train the stacked Bi-RNNs with the feature set $\mathcal{X}$ and ground-truth $\dot{Y}$. Here, MAE is the loss function for stacked Bi-RNNs. An L_2 norm is also introduced to regularize the parameter Ω. The resultant regularized loss is as follows,

$$\mathcal{L}(\Omega) = \frac{1}{NT}\sum_{n=1}^{N}\sum_{t=1}^{T}|\mathbf{z}_t^{(n)} - \dot{\mathbf{y}}_t^{(n)}| + \lambda_R \parallel \Omega \parallel_2, \tag{8}$$

noting that parameter Ω is hidden in $\mathbf{z}$ (see Equation 4).

Mini-batch Stochastic Gradient Descent (SGD) is used to train stacked Bi-RNNs. Adam (Kingma and Ba 2014) is a popular way to train deep neural networks. However, SGD with nesterov momentum (Nesterov 1983) shows a better result if momentum scheduling (Sutskever et al. 2013) is applied. Throughout this study, SGD with nesterov momentum is used as the default optimization method for stacked Bi-RNNs.

2. Initial training of SGCRF Feeding the trained stacked Bi-RNNs with training feature $\mathcal{X}$, the intermediate prediction Z is obtained. Then SGCRF is trained according to Z and ground-truth $\dot{Y}$ by minimizing its negative log-likelihood, which is formulated as

$$L(\Theta, \Lambda) = -\log|\Lambda| + tr(S_{yy}\Lambda + 2S_{yz}\Theta + \Lambda^{-1}\Theta^T S_{zz}\Theta), \tag{9}$$

where S terms are empirical covariance,

$$S_{yy} = \frac{1}{N}\dot{Y}^T\dot{Y}, \quad S_{yz} = \frac{1}{N}\dot{Y}^T Z, \quad S_{zz} = \frac{1}{N}Z^T Z. \tag{10}$$

Equation 9 is a commonly used loss function for SGCRF (Wytock and Kolter 2013)(McCarter and Kim 2016). However, we find that $|\Lambda|$ suffers a risk of overflow when the number of time step is large. Therefore, we introduce eigenvalue decomposition for Λ. As Λ is positive definite, its determinant equals the multiplications of all eigenvalues. Thus, we have

$$\log|\Lambda| = \sum_{i=1}^{T}\log\lambda_i, \tag{11}$$

where λ_i is an eigenvalue. We will use Equation 11 for $\log |\Lambda|$ to avoid the overflow problem.

Recall that it is necessary to apply penalties for sparsity to the parameters, the resultant loss function with L_1 regularization is then as follows,

$$\mathcal{L}(\Theta, \Lambda) = L(\Theta, \Lambda) + \lambda_T \parallel \Theta \parallel_1 + \lambda_L \parallel \Lambda \parallel_{1,*}, \quad (12)$$

where $\lambda_L \parallel \Lambda \parallel_{1,*}$ denotes the L_1 norm of Λ off diagonal elements.

The regularized loss $\mathcal{L}(\Theta, \Lambda)$ is optimized by Newton Coordinate Descent (Newton CD) with active set (Wytock and Kolter 2013). In each iteration, Newton CD finds a generalized Newton descent direction by forming a second-order approximation of the smooth part (i.e. $L(\Theta, \Lambda)$) and minimizes this along with L_1 penalties. Given the Newton direction, the parameters are updates with a step size found by line search. The positive-definite constraint of Λ is taken care of in the line search step.

3. Alternative tuning of stacked Bi-RNNs and SGCRF Stacked Bi-RNNs and SGCRF are then tuned alternatively, using the final loss function $\mathcal{L}(\Omega, \Theta, \Lambda)$

$$\mathcal{L}(\Omega, \Theta, \Lambda) = \mathcal{L}(\Theta, \Lambda) + \lambda_R \parallel \Omega \parallel_2, \quad (13)$$

where the first term is seen in Equation 12, and the second term is the regularization term seen in Equation 8. Recall that parameter Ω is hidden in Z and thus also hidden in $\mathcal{L}(\Theta, \Lambda)$.

Stacked Bi-RNNs are tuned first with the loss function $\mathcal{L}(\Omega, \Theta, \Lambda)$. Θ and Λ are frozen and regarded as constants. The gradient of Ω can be derived according to the chain rule.

$$\frac{\partial \mathcal{L}(\Omega, \Theta, \Lambda)}{\partial \Omega} = \frac{2}{N}(\dot{Y}\Theta^T + Z\Theta\Lambda^{-1}\Theta^T) \cdot \frac{\partial Z}{\partial \Omega} + 2\lambda_R \Omega, \quad (14)$$

where $\partial Z / \partial \Omega$ is handled in the stacked Bi-RNNs. With the obtained gradient, SGD with nesterov momentum can be applied to fine-tune Ω. We feed the fine-tuned stacked Bi-RNNs with training feature $\mathcal{X}$ and obtain new Z. Then, SGCRF is trained according to the new Z and $\dot{Y}$. In this step, Ω is frozen, while Θ and Λ are optimized using Newton CD with active set. The alternated training of stacked Bi-RNNs and SGCRF repeats until they converged. In our study, five alternations are sufficient to reach convergence.

Algorithm 1 summarizes alternative training of CoR.

Algorithm 1 Alternative training of CoR

```
Require: Training feature set $\mathcal{X}$, ground-truth $\dot{Y}$;
Ensure: Parameters $\Omega$, $\Theta$, $\Lambda$;
1: With input $\mathcal{X}$ and $\dot{Y}$, initially train stacked Bi-RNNs
   with respect to the loss function $\mathcal{L}(\Omega)$ (Equation 8);
2: Feed $\mathcal{X}$ to stacked Bi-RNNs to obtain $Z$;
3: With input $Z$ and $\dot{Y}$, initially train SGCRF with respect
   to the loss function $\mathcal{L}(\Theta, \Lambda)$ (Equation 12);
4: while not converged do
5:     Freeze $\Theta$ and $\Lambda$, and train stacked Bi-RNNs with
       input $\mathcal{X}$ and $\dot{Y}$ according to the final loss $\mathcal{L}(\Omega, \Theta, \Lambda)$
       (Equation 13);
6:     Feed $\mathcal{X}$ to the trained stacked Bi-RNNs and obtain
       new $Z$;
7:     Freeze $\Omega$ and train SGCRF with input $Z$ and $\dot{Y}$
       according to $\mathcal{L}(\Omega, \Theta, \Lambda)$;
8: end while
```

End-to-end training

We further propose a more efficient end-to-end training method for CoR. We first set effective initial parameters for CoR, and then fine-tune the parameters end-to-end.

1. Initializations We reuse the first two steps of alternative training to set effective initializations for CoR. Stacked Bi-RNNs are initially trained with respect to the loss function $\mathcal{L}(\Omega)$ (see Equation 8). We feed the trained stacked Bi-RNNs with input data $\mathcal{X}$ to obtain an intermediate prediction Z. Then SGCRF can be trained with Z and $\dot{Y}$ according to the loss $\mathcal{L}(\Theta, \Lambda)$ (see Equation 12). With the initial training of stacked Bi-RNNs and SGCRF, we can set effective initializations for the parameters Ω, Θ and Λ.

The motivation of initialization is as follows. The stacked Bi-RNNs can be naturally trained with a first-order gradient descent method, while the training of SGCRF is much more challenging. Newton CD with active set has been demonstrated as an effective training method for SGCRF, which outperforms other second-order methods (Wytock and Kolter 2013). It is believed that the second-order Newton CD with active set is much more efficient than first-order methods. Consequently, we still employ Newton CD to supply initializations for Θ and Λ. With well initialized parameters, we further use first-order gradient descent to fine-tune the parameters end-to-end.

2. End-to-end fine-tuning We then apply gradient descent to fine-tune the parameters of CoR. However, the parameter Λ is constrained to be positive-definite, and thus the unconstrained gradient descent cannot be directly applied. We resort to the Projected Gradient Descent (PGD) for Λ in the end-to-end training.

PGD comprises two steps. The first step is a regular gradient descent, which updates Λ by $\Lambda^{t+1} = \Lambda^t - \alpha \nabla_\Lambda \mathcal{L}(\Omega, \Theta, \Lambda)$, where $\mathcal{L}(\Omega, \Theta, \Lambda)$ is defined in Equation 13, and α is the learning rate. The second step projects Λ^{t+1} back to the definition domain C of Λ. The projection is defined by $\Pi_C(\Lambda^{t+1}) = argmin_{c \in C} \parallel c - \Lambda^{t+1} \parallel^2$. The projection to a positive definite matrix has been well studied (Henrion and Malick 2012). As Λ^{t+1} is a symmetric matrix, its eigenvalue decomposition can be written as $\Lambda^{t+1} = UDiag[\lambda_1, \cdots, \lambda_T]U^T$, where $Diag[\lambda_1, \cdots, \lambda_T]$ is a diagonal matrix of eigenvalues. The projection is accordingly defined as follows,

$$\Pi_C(\Lambda^{t+1}) \Leftarrow UDiag[max(\mu, \lambda_1), \cdots, max(\mu, \lambda_T)]U^T, \quad (15)$$

where μ is a small positive value.

Λ is updated by PGD, while Ω and Θ are updated by ordinary gradient descent. In implementation, both PGD and gradient descent use a mini-batch mode. In the end-to-end training, SGCRF influences what RNN learns and causes internal covariance shift (Ioffe and Szegedy 2015) in stacked

Bi-RNNs. We find that it is beneficial to apply batch normalization (Ioffe and Szegedy 2015) to the intermediate prediction **z**. We add a batch normalization layer between stacked Bi-RNNs and SGCRF, and thereafter apply end-to-end fine-tuning.

Algorithm 2 summarizes the end-to-end training of CoR.

Algorithm 2 End-to-end training of CoR

```
Require: Training feature set $\mathcal{X}$, ground-truth $\dot{Y}$;
Ensure: Parameters $\Omega$, $\Theta$, $\Lambda$;
1: With input $\mathcal{X}$ and $\dot{Y}$, initially train stacked Bi-RNNs
   with respect to the loss function $\mathcal{L}(\Omega)$ (Equation 8);
2: Feed $\mathcal{X}$ to stacked Bi-RNNs to obtain $Z$;
3: With input $Z$ and $\dot{Y}$, initially train SGCRF with respect
   to the loss function $\mathcal{L}(\Theta, \Lambda)$ (Equation 12);
4: Add a batch normalization layer between stacked Bi-
   RNNs and SGCRF;
5: while $t < max\_iteration$ do
6:     Update $\Omega$, $\Theta$ and $\Lambda$ according to gradient descent
       rule: $\theta^{t+1} = \theta^t - \alpha\nabla_\theta\mathcal{L}(\theta)$, where $\mathcal{L}(\theta)$ is defined
       by Equation 13;
7:     Project $\Lambda^{t+1}$ according to Equation 15;
8:     $t = t + 1$;
9: end while
```

Prediction of CoR

The prediction of CoR comprises two steps.

- Step 1. For a test sample, we feed the observed feature **X** to the stacked Bi-RNNs. Through a feed-forward process, we obtain an intermediate prediction **z**.
- Step 2. We predict **y** given **z** through the prediction process of SGCRF, which seeks to maximize $p(\mathbf{y}|\mathbf{z})$. The underlying model of SGCRF is a Gaussian distribution: $\mathbf{y}|\mathbf{z} = \mathcal{N}(-\mathbf{z}\Theta\Lambda^{-1}, \Lambda^{-1})$. so the maximum of $p(\mathbf{y}|\mathbf{z})$ is the mean of the Gaussian. Thus, the final prediction **y** is formulated as
$$\mathbf{y} = -\mathbf{z}\Theta\Lambda^{-1}. \tag{16}$$

We can see that the prediction of CoR is quite efficient. Due to the Gaussian distribution, we can conveniently derive the 95% confidence interval as follows,

$$\tilde{\mathbf{y}} = \mathbf{y} \pm 1.96 diag(\Lambda^{-1}), \tag{17}$$

where $diag(\Lambda^{-1})$ is the diagonal elements of Λ^{-1}. This equation may assist the decision making in uncertain environments.

Experiment

CoR is evaluated using both synthetic data and real-world data. Experiment 1 evaluates CoR by synthetic data. We demonstrate the advantages of CoR over RNN and SGCRF with synthetic data. Experiment 2 evaluates CoR using a real-world electricity demand prediction competition. In this experiment, CoR shows better performance than state-of-the-art methods in time-series prediction.

In implementations, we use Theano (Theano Development Team 2016) and Lasagne (Dieleman et al. 2015) for deep neural network. For the Newton CD in training SGCRF, we refer to (Wytock and Kolter 2013).

Evaluations using synthetic data

Through the evaluations using synthetic data, we demonstrate the advantages of CoR over SGCRF and stacked Bi-RNNs. As we can change the number of time steps and training samples of synthetic time-series data, we also discover some interesting properties of evaluated models. We furthermore conduct an evaluation using large-scale data to compare the computational cost of SGCRF, stacked Bi-RNNs and CoR.

Evaluations using controlled data The models to be evaluated include SGCRF, stacked Bi-RNNs, RNN+SGCRF (a combination of stacked Bi-RNNs and SGCRF without joint training, which can be defined by Lines 1-3 in Algorithm 1), CoR trained by alternative training (CoR_v1), and CoR trained by end-to-end training (CoR_v2). We change the time steps and sample quantities of synthetic data to evaluate different models, and find some interesting properties of these models.

The synthetic data are generated as follows. The dimension of feature D is fixed as 10, while the time step T can be varied in $\{10, 20, 40\}$, and number of samples N can be varied in $\{1000, 2000, 4000, 8000\}$. The data generation includes three steps:

1) Generate temporally correlated features X. 10 random real values are sampled from a standard Gaussian $\mathcal{N}(0, 1)$, denoted as **x**. The feature $\mathbf{x}_i$ for each time step is generated by $\mathbf{x}_i = (1 - 0.2\mathbf{x}) * \sin(i\pi/T)$, where the sin function models temporal correlations.

2) Introduce nonlinear transformations to features. Each $\mathbf{x}_i$ is transformed by a dense network parameterized by W, which is a $D \times D$ matrix with diagonal $W_{i,i} = 0.8$, sub-diagonal $W_{i-1,i} = -0.2$, super-diagonal $W_{i,i+1} = -0.3$, and the other elements are 0. The following transformation is a dense network parameterized by **v** and $b = 0.3$, where **v** is a D-dimension vector $\mathbf{v}_i = 0.2$. This neural network reduces the dimension of each step output to 1. For both networks, the non-linear activation is leaky ReLU with leakiness = 0.2.

3) Introduce structured information on output. The output of neural network is then put into a SGCRF parameterized by Θ and Λ. Θ is a $T \times T$ diagonal matrix with $\Theta_{i,i} = 0.2$, and Λ is a $T \times T$ matrix with diagonal $\Lambda_{i,i} = 0.8$, sub-diagonal $\Lambda_{i-1,i} = -0.5$ and super-diagonal $\Lambda_{i,i+1} = 0.2$. The output of SGCRF is the final sequence **y** to be predicted.

The simulated multi-step time-series prediction problem has temporal feature correlations, nonlinear feature transformations and structured information. It is adequate to evaluate the performances of different models on time-series prediction. Neural networks and SGCRF are used to encode feature nonlinearities and structured information for convenience, while actually they are equivalent to nonlinear functions.

We evaluate the five models on different time steps with different numbers of samples. In each evaluation, random 80% samples are used for training, while the rest are for testing. Mean Absolute Percentage Error (MAPE) is used as the

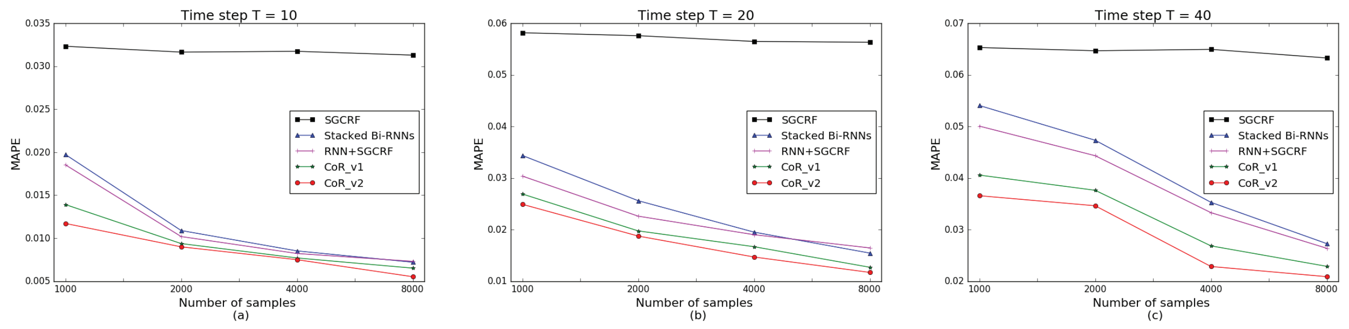

Figure 2: Evaluations of five models on synthetic data. Note that X axes are uneven and the scales of Y axes are different.

metric, which is defined as $|g - \hat{g}|/g \times 100\%$, where g is ground-truth and $\hat{g}$ is prediction. Figure 2 illustrate the evaluation results (best performance is shown for each model with tuned hyper-parameters). We analyze the results in individual figures, and then summarize across three figures.

In Figure 2(a), the time step of time-series data is 10. With different numbers of training samples, the MAPE of SGCRF is relatively stable, about 3.2%. Stacked Bi-RNNs show better results than SGCRF. When the number of training samples increases, the performance of stacked Bi-RNNs improves significantly. This is an indication that deep neural networks favor a large quantity of training data. RNN+SGCRF obtains a little improvement against stacked Bi-RNNs. CoR_v1 and CoR_v2 significantly outperforms the previous three models, and CoR_v2 consistently achieves the best MAPE. It is stressed that when training samples are few, CoR improves stacked Bi-RNNs greatly. This suggests that CoR is a better choice than deep neural networks especially when only a limited number of training samples is available.

In Figure 2(b), the time step of time-series data is 20. For the five models, we observe similar trends in performance variations as those in Figure 2(a). When the time step increases to 40 in Figure 2(c), the observed rules of MAPEs of different models stay the same.

We then summarize across Figure 2(a)-(c). From Figure 2(a) to Figure 2(c), the time step of time-series data is increasing, which indicates that the difficulty of prediction increases. We can see that the performances of different models decrease when the time step increases. SGCRF does not perform well because it misses the nonlinearities in features. Stacked Bi-RNNs outperform SGCRF as they have powerful nonlinearities to fit the data. It is noted that the relative gap of stacked Bi-RNNs and SGCRF decreases when the number of training sample is small (i.e. $N = 1000$). SGCRF is a sparse model that can generalize well with limited training samples, while the performance of stacked Bi-RNNs strongly depends on the number of training samples. A simple combination of stacked Bi-RNNs and SGCRF (i.e. RNN+SGCRF) can improve performance, but not always (see $N = 8000$ in Figure 2 (a) and (b)). CoR can reliably improve MAPE over stacked Bi-RNNs and SGCRF. The end-to-end trained CoR (CoR_v2) shows the best performance. When the training samples are limited, CoR shows significant advantage over other models.

Evaluations using large-scale data We further evaluate SGCRF, stacked Bi-RNNs and CoR on large problems to compare their precision and computation cost. We generate 100K time-series samples with time step $T = 200$ and step feature $D = 100$. The synthetic data are generated in the previous way, while the sizes of parameters are increased accordingly. In this large problem evaluation, we use LSTM to replace standard RNN in stacked Bi-RNNs and CoR, because standard RNN has difficulty in modeling long sequences. CoR is trained by the end-to-end training method, which is more efficient than alternative training. Evaluations are conducted on a server with 8 CPUs and 64 GB Memory. We still use MAPE to measure prediction precisions of different models. Table 1 summarizes the precisions and computation costs of the three models.

Table 1: Comparison of SGCRF, stacked Bi-LSTMs and CoR(LSTM) on a large time-series prediction problem.

Model	MAPE	Memory usage	Time usage
SGCRF	0.065	17.21G	50.33h
stacked Bi-LSTMs	0.032	3.53G	5.43h
CoR(LSTM)	0.026	3.55G	9.41h

In Table 1, the prediction precisions of the three models accord with previous rules. The MAPE of CoR is the lowest among the three models. We pay more attention to the computation cost. The advantages of stacked Bi-LSTMs and CoR over SGCRF come from both modeling and training processes. In modeling time-series, the parameter matrix Θ in SGCRF has a size of $(D \times T) \times T$, while the size of parameter matrix in stacked Bi-LSTMs and CoR is in proportion to $D \times T$. For the same problem, SGCRF has more parameters than stacked Bi-LSTMs and CoR. In training process, the main computation of SGCRF is the calculation of Hessian matrix with a size $(T \times D + T) \times (T \times D + T)$. This a large matrix whose size increases quadratically with respect to T or D. Therefore, when problems grow bigger, SGCRF requires quadratically larger memory to work. For stacked Bi-LSTMs, their memory usage is in proportion to $T \times (T \times D)$, which is much less than that of SGCRF. For CoR(LSTM), there is a little more memory usage than stacked Bi-LSTMs, while its computational time is almost twice as that of stacked Bi-LSTMs.

Evaluations on real-world data

We apply CoR to an electricity demand prediction problem, which is a competition called NPower Forecasting Challenge 2016 [1]. This competition adopted a rolling forecasting mode to simulate the real-world scenario. We follow this mode to evaluate CoR and other comparison models in round one and two (The ground-truth of round three is not available).

The task of this competition is to predict the future power demand in every half hour according to weather data. In round one, the training data range from 2012-01-01 to 2014-09-31, and the task is to predict power demand from 2014-10-01 to 2015-03-01. In round two, the rolling forecasting releases the ground-truth from 2014-10-01 to 2015-03-31, and accordingly the task becomes predicting the power demand from 2015-04-01 to 2015-09-30.

In this time-series prediction problem, the features extracted include the following three categories: 1) Temporal feature, including the year, the month and the number of time step; 2) Calendar feature, including the day of a week and the public holiday; 3) Weather feature, including temperature, cloudiness, altogether nine measurements. Features and ground-truths are normalized by Gaussian (subtracting the mean and dividing by variance), and then input to different models.

We use ARIMA as the baseline model. We also evaluate SGCRF and RNN. SGCRF is evaluated in two modes: SGCRF without feature engineering (SGCRF_w/o) and SGCRF with feature engineering (SGCRF_w). A standard RNN is very hard to train (Note the time step is 48), and thus we evaluate LSTM instead. A two-layer stacked Bi-RNNs can converge steadily, which is evaluated for reference. We also evaluate stacked Bi-LSTMs as comparison. CoR can have two variants, which are based on stacked Bi-RNNs (CoR(RNN)) and stacked Bi-LSTMs (CoR(LSTM)). The two variants are trained by alternative training and end-to-end training. Overall, there are four CoR variants (CoR(RNN)_v1, CoR(RNN)_v2, CoR(LSTM)_v1, CoR(LSTM)_v2) to evaluate. Moreover, we evaluate other two state-of-the-art methods, namely, Gradient boosting (Chen and Guestrin 2016) with feature engineering (add previous time step features to the current), and attentional stacked Bi-LSTMs (Bahdanau, Cho, and Bengio 2014).

Unless clearly noted with feature engineering, all models are evaluated on the raw features. This competition uses MAPE as the evaluation metric. Table 2 summarizes the evaluation results of different models. The top three results in this competition are listed for reference. These winning methods did not employ sophisticated models, but were concerned with detailed features and feature engineering [2]. The basic ARIMA model achieves MAPE of 8.95% and 8.77%, which are not as good as the top three results in the competition.

The following methods are very recent. SGCRF_w/o is almost as competitive as the 2nd Place, while SGCRF_w shows significantly improvement compared to SGCRF_w/o. Models based on deep neural networks outperform SGCRF. The overall performance of LSTM is comparable to the 1st Place. Stacked Bi-RNNs are slightly better than a single LSTM. In contrast, stacked Bi-LSTMs achieve lower MAPE than stacked Bi-RNNs. It suggests that LSTM is more effective than standard RNN when the number of time step gets larger. Even though we apply feature engineering to gradient boosting, it does not show any advantage over deep RNN models. The state-of-the-art attentional LSTM (also in a stacked bi-directional structure) slightly outperforms stacked Bi-LSTMs.

Table 2: Evaluation results of the two rounds in NPower Forecasting Challenge 2016.

Model	MAPE in round 1	MAPE in round 2
1st Place	3.14%	7.13%
2nd Place	6.43%	4.89%
3rd Place	7.84%	7.48%
ARIMA	8.95%	8.77%
SGCRF_w/o	5.83%	6.64%
SGCRF_w	4.91%	5.60%
LSTM	(4.92 ± 0.15)%	(4.73 ± 0.17)%
stacked Bi-RNNs	(4.88 ± 0.21)%	(4.56 ± 0.20)%
stacked Bi-LSTMs	(4.43 ± 0.16)%	(4.31 ± 0.14)%
Gradient boosting	4.90%	4.61%
Attentional LSTM	(4.37 ± 0.15)%	(4.25 ± 0.16)%
CoR(RNN)_v1	(4.25 ± 0.04)%	(4.10 ± 0.05)%
CoR(RNN)_v2	(4.19 ± 0.06)%	(4.03 ± 0.05)%
CoR(LSTM)_v1	(4.11 ± 0.04)%	(3.94 ± 0.03)%
CoR(LSTM)_v2	(4.05 ± 0.05)%	(3.87 ± 0.03)%

The four variants of CoR show better results compared to previous models. In building different CoR variants, we use deep RNN models which achieve median performances (note the fluctuations in results). CoR(RNN)_v2 achieves an average MAPE of 4.11%, which is relative 10.8% improvement on stacked Bi-RNNs. Similarly, CoR(LSTM)_v2 achieves relative 9.8% improvement on stacked Bi-LSTMs. Moreover, CoR models are more stable than deep RNNs. The best model, CoR(LSTM)_v2 achieves an average MAPE of 3.96%, which is much better than 5.14%, the average MAPE of 1st Place. The disadvantage of CoR and deep RNN models is that the result suffers fluctuations, but this can be compensated by model ensemble.

Conclusion

This paper proposes CoR, which integrates the advantages of RNN and SGCRF, for multi-step time-series prediction. Two training methods are proposed for CoR. Experimental results show that the end-to-end training is more efficient than the alternative training. The evaluations with both synthetic data and real-world data demonstrate that CoR can significantly improve prediction precision over stacked Bi-RNNs and SGCRF. CoR also outperforms other state-of-the-art methods in the real-world multi-step time-series prediction. The success of CoR suggests that both nonlinear and temporal correlations in observed data and structured output information are important in multi-step time-series prediction.

[1] https://www.npowerjobs.com/graduates/forecasting-challenge. Data are publicly available. Competition results are also published on this webpage.

[2] http://blog.drhongtao.com/2016/12/winning-methods-from-npower-forecasting-challenge-2016.html

Acknowledgments

The authors would like to thank Lei Wang and Luping Zhou for their helpful discussion. This work is supported by a Discovery Project (DP140100974) from Australian Research Council.

References

Bahdanau, D.; Cho, K.; and Bengio, Y. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bengio, Y., et al. 2009. Learning deep architectures for ai. *Foundations and trends® in Machine Learning* 2(1):1–127.

Box, G. E., and Pierce, D. A. 1970. Distribution of residual autocorrelations in autoregressive-integrated moving average time series models. *Journal of the American statistical Association* 65(332):1509–1526.

Box, G. E.; Jenkins, G. M.; Reinsel, G. C.; and Ljung, G. M. 2015. *Time series analysis: forecasting and control*. John Wiley & Sons.

Chen, T., and Guestrin, C. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 785–794. ACM.

Dieleman, S.; Schlter, J.; Raffel, C.; Olson, E.; Snderby, S. K.; Nouri, D.; Maturana, D.; Thoma, M.; Battenberg, E.; Kelly, J.; Fauw, J. D.; Heilman, M.; de Almeida, D. M.; McFee, B.; Weideman, H.; Takcs, G.; de Rivaz, P.; Crall, J.; Sanders, G.; Rasul, K.; Liu, C.; French, G.; and Degrave, J. 2015. Lasagne: First release.

Friedman, J. H. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* 1189–1232.

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

Henrion, D., and Malick, J. 2012. Projection methods in conic optimization. In *Handbook on Semidefinite, Conic and Polynomial Optimization*. Springer. 565–600.

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

Ioffe, S., and Szegedy, C. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 448–456.

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

Lample, G.; Ballesteros, M.; Subramanian, S.; Kawakami, K.; and Dyer, C. 2016. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*.

Lin, T.; Guo, T.; and Aberer, K. 2017. A dual-stage attention-based recurrent neural network for time series prediction. In *IJCAI*.

Lin, C.-J. 2007. Projected gradient methods for nonnegative matrix factorization. *Neural computation* 19(10):2756–2779.

McCarter, C., and Kim, S. 2016. Large-scale optimization algorithms for sparse conditional gaussian graphical models. In *Artificial Intelligence and Statistics*, 528–537.

Nesterov, Y. 1983. A method of solving a convex programming problem with convergence rate o (1/k2). In *Soviet Mathematics Doklady*, volume 27, 372–376.

Osogami, T., and Otsuka, M. 2015. Learning dynamic boltzmann machines with spike-timing dependent plasticity. *arXiv preprint arXiv:1509.08634*.

Pascanu, R.; Gulcehre, C.; Cho, K.; and Bengio, Y. 2013. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*.

Pineda, F. J. 1987. Generalization of back-propagation to recurrent neural networks. *Physical review letters* 59(19):2229.

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

Theano Development Team. 2016. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints* abs/1605.02688.

Wytock, M., and Kolter, Z. 2013. Sparse gaussian conditional random fields: Algorithms, theory, and application to energy forecasting. In *International conference on machine learning*, 1265–1273.

Zheng, S.; Jayasumana, S.; Romera-Paredes, B.; Vineet, V.; Su, Z.; Du, D.; Huang, C.; and Torr, P. H. 2015. Conditional random fields as recurrent neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, 1529–1537.