A Tractable Model for High-Dimensional Real Sequences

Anonymous Author(s)
Affiliation
Address
email

Abstract

We present a probabilistic model for generating and modelling real-valued high dimensional sequences. This model combines a powerful distribution estimator, the Real-Valued Neural Autoregressive Density Estimator (RNADE) with a Recurrent Neural Network (RNN) for capturing temporal dependencies in sequences of high dimensional data. Maximum likelihood learning can be applied to efficiently train the model using a gradient based optimiser. Unlike other models in this family, log-likelihoods for sequences can be computed exactly and efficiently. We evaluate the model's performance on standard datasets and compare its performance to other models.

1 Introduction

Modeling sequences is a fundamental problem in machine learning. Speech, music, video and many other kinds of data are sequential in nature. A good sequential model can be used for discriminative tasks, for sequence completion, as a prior in more complex tasks, sequence denoising and various other applications. There are many existing models such as Linear Dynamical Systems and Hidden Markov Models (HMMs) that model sequential data. Although HMMs are widely used in speech recognition and a wide variety of other problems, they are limited by the exact form of their latent representation and become computationally infeasible for modelling long histories [9].

Recurrent Neural Networks (RNNs) are simple and powerful models for sequential data. RNNs have an internal memory that allows them to model dependencies between observations occurring at different times. RNNs are completely general in their description in that in principle they can describe relationships between inputs separated by arbitrary lengths of time. However in practice, training RNNs using gradient based optimisers is not easy and their use in practical applications has been limited. Recently RNNs have received a lot of attention from the machine learning community and various problems that plagued gradient based training have been identified and some of them have been effectively remedied [6, 2]. RNNs have been shown to be very successful on several music and language applications [7, 4, 2].

An RNN can be used to map an input sequence to an output sequence or it can be used as a generative model. Generative models try to extrapolate a finite number of training points to cover the entire space of inputs. In its simplest form, the RNN can predict unimodal outputs under the assumption that the output variables are independent. However for many problems this is not satisfactory. A more powerful RNN-based generative model can be constructed by letting the RNN predict the parameters of a powerful distribution estimator. By doing this, we can get complex high-dimensional distributions at each time-step conditioned on the previous inputs. This idea was first employed in the Recurrent Temporal Restricted Boltzmann Machine (RTRBM) [10] model. The model was then extended by combining an RNN with a Neural Autoregressive Distribution Estimator (NADE) and

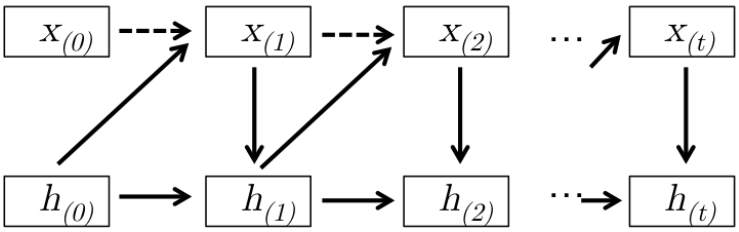

Figure 1: Awesome RNN

the RBM via a more general architecture [4]. The RNN-RBM and the RNN-NADE have been successfully used for tasks in speech and music [5, 4].

Although the models described above have been successful in various tasks, they are unsuitable for modelling real-valued data. Both the RBM and the NADE explicitly model binary vectors. The extension of the RBM for modelling real-valued data using either the mean-field approximation or using gaussian visible units is known to have several limitations [5]. In this paper, we present a new model for modelling real-valued sequential data by combining the RNN with the real-valued neural autoregressive density estimator (RNADE). The RNADE is a density estimator that has been shown to outperform gaussian mixture models (GMMs) and a host of other density estimators on several tasks [11]. We show that obtaining probabilities and log-likelihoods from the combined RNN-RNADE model is tractable and fast. We show that the gradients with respect to the model parameters can be calculated exactly and that training can be easily performed with a gradient based optimiser. We evaluate the model's performance on real-valued datasets and demonstrate a marked improvement in performance.

2 Recurrent Neural Networks as generative models

An RNN defines a distribution over an output sequence $\mathbf{x} \equiv \{\mathbf{x}^t \in \mathbb{R}^n, t \leq T\}$ in the following manner:

$$P(\mathbf{x}) = \prod_{t=1}^{T} P(\mathbf{x}^t | \mathcal{A}^t)$$

where $\mathcal{A}^t \equiv \{\mathbf{x}^\tau | \tau < t\}$ is the history of the input sequence $\mathbf{x}$ so far, $P(\mathbf{x}^t|\mathcal{A}^t)$ is probability of observing observing $\mathbf{x}^t$ conditioned on the history of the sequence $\mathcal{A}^t$ at time t. RNN based generative models with different properties and capacities can be constructed by carefully choosing the form and parameterization of the conditional $P(\mathbf{x}^t|\mathcal{A}^t)$ as discussed in this section.

In an RNN with a single hidden layer, the hidden layer state at time t is given by:

$$\mathbf{h}^t = \sigma(W_{in}\mathbf{x}^t + W_{rec}\mathbf{h}^{t-1} + \mathbf{b}_h) \tag{1}$$

where W_{in} is the weight matrix from the input to the hidden layer, W_{rec} is the weight matrix from the previous hidden state to the current state and $\mathbf{b}_h$ is the bias vector for the hidden layer. In a standard RNN, the next time step $\mathbf{x}^{t+1}$ is predicted as:

$$\mathbf{x}^{t+1} = \sigma(W_{out}\mathbf{h}^t + \mathbf{b}_{out})$$

where W_{out} connects the hidden layer to the output layer and $\mathbf{b}_{out}$ is the bias. As shown in Figure 1, there can be optional connections between the input and the output and the output and the hidden layers for temporal smoothing. However as mentioned earlier, each of the outputs is unimodal and independent of the other outputs. These assumptions are not true for many types of data and RNNs that can predict high-dimensional multi-model conditional distributions are necessary. This can be achieved by letting the hidden layer of the RNN predict some of the parameters of a distribution estimator instead of predicting the outputs directly.

In previous work [4], the RNN was used to predict the hidden and visible biases of the an RBM and a NADE model to yield complex conditional distributions at each time step. If the gradients of a suitable cost function can be obtained with respect to the parameters of the distribution estimator, then the entire model can be trained using back-propagation through time [8]. Although these

models are very good at modelling sequences of binary vectors, they are unsuitable for modelling real-valued data. In order to use the power of this architecture for modelling real data, we present a model by combining the RNN and the RNADE. The RNADE is discussed in detail in following section and combination of the two models is described in section 4.

3 The RNADE

The RNADE is a generalisation of the NADE to real-valued data. Like the NADE, the RNADE expresses the joint probability of the data, as a product of one-dimensional conditional distributions as follows:

$$p(x) = \prod_{d=1}^{D} p(x_d|\mathbf{x}_{<\mathbf{d}}) \text{ with } p(x_d|\mathbf{x}_{<\mathbf{d}}) = p_{\mathcal{M}(x_d|\theta_d)}$$

where $p_{\mathcal{M}}$ is a mixture of Gaussians and $\boldsymbol{x}_{<d}$ is a vector of all the dimensions of the data point $< d$. The RNADE is computationally efficient because of the weight sharing employed in the calculation of the hidden state:

$$\mathbf{a}_d = \boldsymbol{W}_{.,<d}\boldsymbol{x}_d + \mathbf{c}$$
$$\boldsymbol{h}_d = \sigma(\rho_d \mathbf{a}_d)$$

where $\mathbf{c} \in \mathbb{R}^H$ and $\boldsymbol{W} \in \mathbb{R}^{D\times(H-1)}$ are neural network parameters that are shared across all the neural networks and $\sigma(x) = 1/(1+e^{-x})$ is the sigmoid function. $\boldsymbol{W}_{.,<d}$ represents the first $D-1$ columns of the shared weight matrix. The term ρ_d is a scaling factor which is also learnt from the data. The scaling factor was introduced in [1] in order to prevent the sigmoid hidden units from saturating. The computation of the activations of the hidden units can be made more efficient by performing the computation as:

$$\mathbf{a}_1 = \mathbf{c}, \quad \mathbf{a}_{d+1} = \mathbf{a}_d + x_d \mathbf{W}_{.,d}$$

Unlike the NADE which models each output as a bernouilli distribution, the outputs of each of the feed forward neural networks of the RNADE are mixtures of Gaussians. Therefore the RNADE comprises of D mixture density networks with tied input-to-hidden weights. Once the hidden units of the RNADE have been computed, they are used to compute the parameters of the GMMs $\boldsymbol{\theta}_d = \{\boldsymbol{\alpha}_d, \boldsymbol{\mu}_d, \boldsymbol{\sigma}_d\}$ at each output, where $\boldsymbol{\alpha}_d$ are the mixing coefficients, $\boldsymbol{\mu}_d$ are the means and $\boldsymbol{\sigma}_d$ are the variances. These parameters are computed as follows:

$$\boldsymbol{\alpha}_d = \text{softmax}({\mathbf{V}_d^{\alpha}}^T \mathbf{h}_d + \mathbf{b}_d^{\alpha})$$
$$\boldsymbol{\mu}_d = {\mathbf{V}_d^{\mu}}^T \mathbf{h}_d + \mathbf{b}_d^{\mu}$$
$$\boldsymbol{\sigma}_d = \exp({\mathbf{V}_d^{\sigma}}^T \mathbf{h}_d + \mathbf{b}_d^{\sigma})$$

where $\mathbf{V}_d^{\alpha}, \mathbf{V}_d^{\mu}, \mathbf{V}_d^{\sigma}$ are $H \times K$ matrices , $\mathbf{b}_d^{\alpha}, \mathbf{b}_d^{\mu}, \mathbf{b}_d^{\sigma}$ are vectors of size K and K is the number of components in the GMM. More concisely, the RNADE is parameterised by $\mathbf{V}^{\alpha}, \mathbf{V}^{\mu}, \mathbf{V}^{\sigma}$ which are $D \times H \times K$ matrices and $\mathbf{b}^{\alpha}, \mathbf{b}^{\mu}, \mathbf{b}^{\sigma}$ which are matrices of size $D \times K$. The parameters of the RNADE can be learnt by performing gradient ascent on the log-likelihood of the training set.

4 RNN-RNADE

The RNN-RNADE is a sequence of conditional distributions for each time-step of a sequence $\mathbf{x}$. As mentioned earlier, the parameters of the conditional distributions at each time step t are a function of the hidden state of the RNN at the previous time-step $t-1$. We first define the notation in order to simplify discussion of the training algorithm later in the section.

From section 3, the RNADE is parameterised by $\theta \equiv \{\mathbf{V}^{\alpha}, \mathbf{V}^{\mu}, \mathbf{V}^{\sigma}, \mathbf{b}^{\alpha}, \mathbf{b}^{\mu}, \mathbf{b}^{\sigma}\}$. Let $\theta_t \equiv \{\mathbf{V}_t^{\alpha}, \mathbf{V}_t^{\mu}, \mathbf{V}_t^{\sigma}, \mathbf{b}_t^{\alpha}, \mathbf{b}_t^{\mu}, \mathbf{b}_t^{\sigma}\}$ denote the parameters of the conditional RNADE at time t. Although all the parameters of the RNADE at time t can be a function of the hidden state at $t-1$, we consider the matrices $\mathbf{V}^{\alpha}, \mathbf{V}^{\mu}, \mathbf{V}^{\sigma}$ to be fixed at each time step and only consider the case

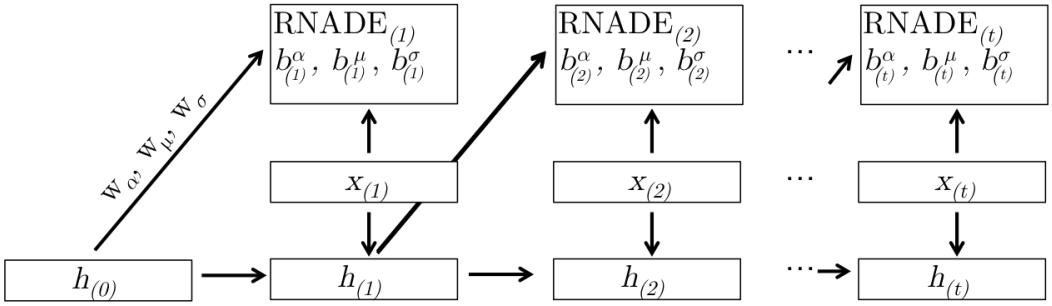

Figure 2: Awesome RNN

where the biases $\mathbf{b}^\alpha, \mathbf{b}^\mu, \mathbf{b}^\sigma$ to be time-dependent through the RNN hidden state. Therefore, $\theta_t \equiv \{\mathbf{V}^\alpha, \mathbf{V}^\mu, \mathbf{V}^\sigma, \mathbf{b}_t^\alpha, \mathbf{b}_t^\mu, \mathbf{b}_t^\sigma\}$. As described later, we experiment with the architecture to ascertain which permutation of time dependent parameters $\mathbf{b}_t^\alpha, \mathbf{b}_t^\mu, \mathbf{b}_t^\sigma$ gives the best results.

As illustrated by figure 2, the time-dependent RNADE parameters are give by:

$$\text{rvec}(\mathbf{b}_t^\alpha) = \text{rvec}(\mathbf{b}^\alpha) + W_\alpha \mathbf{h}^{t-1} \tag{2}$$

$$\text{rvec}(\mathbf{b}_t^\mu) = \text{rvec}(\mathbf{b}^\mu) + W_\mu \mathbf{h}^{t-1} \tag{3}$$

$$\text{rvec}(\mathbf{b}_t^\sigma) = \text{rvec}(\mathbf{b}^\sigma) + W_\sigma \mathbf{h}^{t-1} \tag{4}$$

where $\text{rvec}(A) = [a_{1,1}, ..., a_{1,n}, a_{2,1}, ..., a_{2,n}, ..., a_{m,1}, ..., a_{m,n}]$ for some $m \times n$ matrix A, $W_\alpha, W_\mu, W_\sigma \in \mathbb{R}^{r \times Dc}$ are matrices from the hidden state of the RNN to the mixing coefficients, means and standard deviations of the RNADE respectively, r is the number of hidden units in the RNN, D is the dimensionality of the input vectors and c is the number of components of each GMM in the RNADE.

Therefore using equations 2,3,4 we can obtain the time-dependent biases for the RNADE at each time step.

4.1 Learning in the RNN-RNADE

For training the RNN-RNADE, we minimise the negative log-likelihood cost of observing a sequence $\mathbf{x}$. The cost function is given by:

$$\begin{aligned} L(\theta) &= -\log P(\mathbf{x}) \\ &= -\sum_{t=1}^{T} \log P(\mathbf{x}^t; \theta^t) \end{aligned}$$

The gradients of the conditionals $P(\mathbf{x}^t; \theta^t)$ with respect to the parameters of the RNADE can be calculated, the details of which are outlined in [11]. Once we obtain the gradients with respect to the time-dependent parameters, the entire network can be trained using BPTT.

Suppose we obtain $\frac{\partial L}{\partial \mathbf{b}_t^\alpha}, \frac{\partial L}{\partial \mathbf{b}_t^\mu}, \frac{\partial L}{\partial \mathbf{b}_t^\sigma}$, then the gradients with respect to the other model parameters can be calculated as follows:

$$\frac{\partial L}{\partial W_\alpha} = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{b}_t^\alpha} \mathbf{h}^{t-1} \tag{5}$$

$$\frac{\partial L}{\partial W_\mu} = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{b}_t^\mu} \mathbf{h}^{t-1} \tag{6}$$

$$\frac{\partial L}{\partial W_\sigma} = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{b}_t^\sigma} \mathbf{h}^{t-1} \tag{7}$$

Using equations 1,2,3,4:

$$\frac{\partial L}{\partial \mathbf{h}^t} = W_{rec} \frac{\partial L}{\partial \mathbf{h}^{t+1}} \mathbf{h}^{t+1}(1 - \mathbf{h}^{t+1}) + W_\alpha \frac{\partial L}{\partial \mathbf{b}_{t+1}^\alpha} + W_\mu \frac{\partial L}{\partial \mathbf{b}_{t+1}^\mu} + W_\sigma \frac{\partial L}{\partial \mathbf{b}_{t+1}^\sigma} \tag{8}$$

Using equation 8, the gradients with respect to the remaining model parameters can be calculated easily.

$$\frac{\partial L}{\partial \mathbf{b}_h} = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{h}^t} \mathbf{h}^t(1 - \mathbf{h}^t)$$

$$\frac{\partial L}{\partial W_{rec}} = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{h}^t} \mathbf{h}^t(1 - \mathbf{h}^t)\mathbf{h}^{t-1}$$

$$\frac{\partial L}{\partial W_{in}} = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{h}^t} \mathbf{h}^t(1 - \mathbf{h}^t)\mathbf{x}^t$$

As shown above, the gradients of the cost function can be calculated and used to train the entire model. In our implementation of the model, we used the automatic differentiation library Theano [3] instead of calculating the gradients by hand. In the supplementary material, we show detailed calculations for the gradient calculations. Unlike the RNN-RBM, the gradients of the RNN-RNADE model can be calculated exactly. Therefore model training can benefit from the use of more powerful gradient based optimiser like the Hessian Free optimiser [6]

5 Experiments

The RNN-RNADE is tested on three tasks, a simple 2d trajectory, videos of bouncing balls and motion capture data. On all three tasks we report the log probabilities as well as the squared prediction error, allowing comparisons between our approach and the RTRBM and RNN-RBM in the latter two tasks. Code for experiments is available in the supplementary material.

5.1 2d stochastic trajectory

This task was used as a demonstrative example of the system. At each time step a particles [x,y] coordinates are determined by the system of equations:

$x = N(sin(t) + 1, 0.1)$
$y = x.N(sin(t + 0.5) + 1, 0.2)$

where t is the timestep. The model is trained on sequences of 100 consecutive timesteps. A model was trained consisting of an RNADE with 1 component per dimension and 20 hidden neurons per dimension, with the RNN employing 20 recurrent neurons. The squared prediction error per frame is 0.16. Videos of predictions of the next time frame, as well as sequence completion can be seen in the supplementary material. A limited grid search was performed to find hyperparameters such as number of hidden and recurrent neurons,

5.2 Motion capture data

As in [10] and [4] we use the motion capture dataset from [10], which consists of sequences of 50 frames, each containing 49 real-valued dimensions representing joint angles, translations and rotations of the base of the spine. Using an RNADE with 1 component per dimension and 200 hidden neurons per dimension and 200 recurrent neurons in the RNN, the RNN-RNADE produces a mean squared test error per frame of 2, substantially lower than 20.1 for RTRBM and 16.2 for the RNN-RBM as reported in [4].

5.3 Videos of bouncing balls

The bouncing balls dataset consists of synthetic videos of three balls bouncing in a box, as described in [10]. We use videos of 15x15 pixels (225 dimensions) and sequences of 128 frames. Each pixel is a real value in the range [0,1]. As the RNADE samples from a Gaussian distribution, it is possible for illegal values to be produced and so the output of the model is constrained to the minimum and maximum values of the data. Using an RNADE with 1 component per dimension and 200 hidden neurons per dimension and 200 recurrent neurons in the RNN, the RNN-RNADE achieves a squared test error per frame of 2.0, similar to the RTRBM but substantially higher than the RNN-RBM as reported in [4]. Videos of errors per frame and generated sequences can be found in the supplementary material.

References

[1] Yoshua Bengio. Discussion of "The Neural Autoregressive Distribution Estimator". In Geoffrey J. Gordon and David B. Dunson, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11)*, volume 15, pages 38–39. Journal of Machine Learning Research - Workshop and Conference Proceedings, 2011.

[2] Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. Advances in optimizing recurrent networks. 2012.

[3] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

[4] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In *29th International Conference on Machine Learning*, Edinburgh, Scotland, UK, 2012.

[5] Nicolas Boulanger-Lewandowski, Jasha Droppo, Mike Seltzer, and Dong Yu. Phone sequence modeling with recurrent neural networks. In *Proceedings of the 2014 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2014), Florence, Italy*, May 2014.

[6] James Martens and Ilya Sutskever. Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1033–1040, 2011.

[7] Tomas Mikolov, Anoop Deoras, Stefan Kombrink, Lukas Burget, and Jan Cernockỳ. Empirical evaluation and combination of advanced language modeling techniques. In *INTERSPEECH*, pages 605–608, 2011.

[8] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[9] Ilya Sutskever and Geoffrey E Hinton. Learning multilevel distributed representations for high-dimensional sequences. In *International Conference on Artificial Intelligence and Statistics*, pages 548–555, 2007.

[10] Ilya Sutskever, Geoffrey E Hinton, and Graham W Taylor. The recurrent temporal restricted boltzmann machine. In *Advances in Neural Information Processing Systems*, pages 1601–1608, 2008.

[11] Benigno Uria, Iain Murray, and Hugo Larochelle. Rnade: The real-valued neural autoregressive density-estimator. In *Advances in Neural Information Processing Systems 26*, pages 2175–2183. 2013.