Journal of Machine Learning Research 17 (2016) 1-37 Submitted 5/16; Published 9/16

Neural Autoregressive Distribution Estimation

Benigno Uria BENIGNO.URIA@GMAIL.COM
Google DeepMind
London, UK

Marc-Alexandre Côté MARC-ALEXANDRE.COTE@USHERBROOKE.CA
Department of Computer Science
Université de Sherbrooke
Sherbrooke, J1K 2R1, QC, Canada

Karol Gregor KAROL.GREGOR@GMAIL.COM
Google DeepMind
London, UK

Iain Murray I.MURRAY@ED.AC.UK
School of Informatics
University of Edinburgh
Edinburgh EH8 9AB, UK

Hugo Larochelle HLAROCHELLE@TWITTER.COM
Twitter
141 Portland St, Floor 6
Cambridge MA 02139, USA

Editor: Ruslan Salakhutdinov

Abstract

We present Neural Autoregressive Distribution Estimation (NADE) models, which are neural network architectures applied to the problem of unsupervised distribution and density estimation. They leverage the probability product rule and a weight sharing scheme inspired from restricted Boltzmann machines, to yield an estimator that is both tractable and has good generalization performance. We discuss how they achieve competitive performance in modeling both binary and real-valued observations. We also present how deep NADE models can be trained to be agnostic to the ordering of input dimensions used by the autoregressive product rule decomposition. Finally, we also show how to exploit the topological structure of pixels in images using a deep convolutional architecture for NADE.

Keywords: deep learning, neural networks, density modeling, unsupervised learning

1. Introduction

Distribution estimation is one of the most general problems addressed by machine learning. From a good and flexible distribution estimator, in principle it is possible to solve a variety of types of inference problem, such as classification, regression, missing value imputation, and many other predictive tasks.

Currently, one of the most common forms of distribution estimation is based on directed graphical models. In general these models describe the data generation process as sampling

a latent state $\boldsymbol{h}$ from some prior $p(\boldsymbol{h})$, followed by sampling the observed data $\boldsymbol{x}$ from some conditional $p(\boldsymbol{x} \,|\, \boldsymbol{h})$. Unfortunately, this approach quickly becomes intractable and requires approximations when the latent state $\boldsymbol{h}$ increases in complexity. Specifically, computing the marginal probability of the data, $p(\boldsymbol{x}) = \sum_{\boldsymbol{h}} p(\boldsymbol{x} \,|\, \boldsymbol{h})\, p(\boldsymbol{h})$, is only tractable under fairly constraining assumptions on $p(\boldsymbol{x} \,|\, \boldsymbol{h})$ and $p(\boldsymbol{h})$.

Another popular approach, based on undirected graphical models, gives probabilities of the form $p(\boldsymbol{x}) = \exp\{\phi(\boldsymbol{x})\}/Z$, where ϕ is a tractable function and Z is a normalizing constant. A popular choice for such a model is the restricted Boltzmann machine (RBM), which substantially out-performs mixture models on a variety of binary data sets (Salakhutdinov and Murray, 2008). Unfortunately, we often cannot compute probabilities $p(\boldsymbol{x})$ exactly in undirected models either, due to the normalizing constant Z.

In this paper, we advocate a third approach to distribution estimation, based on autoregressive models and feed-forward neural networks. We refer to our particular approach as Neural Autoregressive Distribution Estimation (NADE). Its main distinguishing property is that computing $p(\boldsymbol{x})$ under a NADE model is tractable and can be computed efficiently, given an arbitrary ordering of the dimensions of $\boldsymbol{x}$.

The NADE framework was first introduced for binary variables by Larochelle and Murray (2011), and concurrent work by Gregor and LeCun (2011). The framework was then generalized to real-valued observations (Uria et al., 2013), and to versions based on deep neural networks that can model the observations in any order (Uria et al., 2014). This paper pulls together an extended treatement of these papers, with more experimental results, including some by Uria (2015). We also report new work on modeling 2D images by incorporating convolutional neural networks into the NADE framework. For each type of data, we're able to reach competitive results, compared to popular directed and undirected graphical model alternatives.

2. NADE

We consider the problem of modeling the distribution $p(\boldsymbol{x})$ of input vector observations $\boldsymbol{x}$. For now, we will assume that the dimensions of $\boldsymbol{x}$ are binary, that is $x_d \in \{0, 1\}\ \forall d$. The model generalizes to other data types, which is explored later (Section 3) and in other work (Section 8).

NADE begins with the observation that any D-dimensional distribution $p(\boldsymbol{x})$ can be factored into a product of one-dimensional distributions, in any order o (a permutation of the integers $1, \ldots, D$):

$$p(\boldsymbol{x}) = \prod_{d=1}^{D} p(x_{o_d} \,|\, \boldsymbol{x}_{o_{<d}}). \tag{1}$$

Here $o_{<d}$ contains the first $d-1$ dimensions in ordering o and $\boldsymbol{x}_{o_{<d}}$ is the corresponding subvector for these dimensions. Thus, one can define an 'autoregressive' generative model of the data simply by specifying a parameterization of all D conditionals $p(x_{o_d} \,|\, \boldsymbol{x}_{o_{<d}})$.

Frey et al. (1996) followed this approach and proposed using simple (log-)linear logistic regression models for these conditionals. This choice yields surprisingly competitive results, but are not competitive with non-linear models such as an RBM. Bengio and Bengio (2000) proposed a more flexible approach, with a single-layer feed-forward neural network for each

conditional. Moreover, they allowed connections between the output of each network and the hidden layer of networks for the conditionals appearing earlier in the autoregressive ordering. Using neural networks led to some improvements in modeling performance, though at the cost of a really large model for very high-dimensional data.

In NADE, we also model each conditional using a feed-forward neural network. Specifically, each conditional $p(x_{o_d} \mid \boldsymbol{x}_{<d})$ is parameterized as follows:

$$p(x_{o_d} = 1 \mid \boldsymbol{x}_{o_{<d}}) = \text{sigm}\left(\boldsymbol{V}_{o_d,\cdot}\boldsymbol{h}_d + b_{o_d}\right) \tag{2}$$

$$\boldsymbol{h}_d = \text{sigm}\left(\boldsymbol{W}_{\cdot,o_{<d}}\boldsymbol{x}_{o_{<d}} + \boldsymbol{c}\right), \tag{3}$$

where $\text{sigm}(a) = 1/(1+e^{-a})$ is the logistic sigmoid, and with H as the number of hidden units, $\boldsymbol{V} \in \mathbb{R}^{D\times H}$, $\boldsymbol{b} \in \mathbb{R}^D$, $\boldsymbol{W} \in \mathbb{R}^{H\times D}$, $\boldsymbol{c} \in \mathbb{R}^H$ are the parameters of the NADE model.

The hidden layer matrix $\boldsymbol{W}$ and bias $\boldsymbol{c}$ are shared by each hidden layer $\boldsymbol{h}_d$ (which are all of the same size). This parameter sharing scheme (illustrated in Figure 1) means that NADE has $O(HD)$ parameters, rather than $O(HD^2)$ required if the neural networks were separate. Limiting the number of parameters can reduce the risk of over-fitting. Another advantage is that all D hidden layers $\boldsymbol{h}_d$ can be computed in $O(HD)$ time instead of $O(HD^2)$. Denoting the pre-activation of the d^{th} hidden layer as $\boldsymbol{a}_d = \boldsymbol{W}_{\cdot,o_{<d}}\boldsymbol{x}_{o_{<d}} + \boldsymbol{c}$, this complexity is achieved by using the recurrence

$$\boldsymbol{h}_1 = \text{sigm}\left(\boldsymbol{a}_1\right), \quad \text{where } \boldsymbol{a}_1 = \boldsymbol{c} \tag{4}$$

$$\boldsymbol{h}_d = \text{sigm}\left(\boldsymbol{a}_d\right), \quad \text{where } \boldsymbol{a}_d = \boldsymbol{W}_{\cdot,o_{<d}}\boldsymbol{x}_{o_{<d}} + \boldsymbol{c} = \boldsymbol{W}_{\cdot,o_{d-1}}x_{o_{d-1}} + \boldsymbol{a}_{d-1} \quad \text{for } d \in \{2,\ldots,D\}, \tag{5}$$

where Equation 5 given vector $\boldsymbol{a}_{d-1}$ can be computed in $O(H)$. Moreover, the computation of Equation 2 given $\boldsymbol{h}$ is also $O(H)$. Thus, computing $p(\boldsymbol{x})$ from D conditional distributions (Equation 1) costs $O(HD)$ for NADE. This complexity is comparable to that of regular feed-forward neural network models.

NADE can be trained by maximum likelihood, or equivalently by minimizing the average negative log-likelihood,

$$\frac{1}{N}\sum_{n=1}^{N} -\log p(\boldsymbol{x}^{(n)}) = \frac{1}{N}\sum_{n=1}^{N}\sum_{d=1}^{D} -\log p(x_{o_d}^{(n)} \mid \boldsymbol{x}_{o_{<d}}^{(n)}), \tag{6}$$

usually by stochastic (minibatch) gradient descent. As probabilities $p(\boldsymbol{x})$ cost $O(HD)$, gradients of the negative log-probability of training examples can also be computed in $O(HD)$. Algorithm 1 describes the computation of both $p(\boldsymbol{x})$ and the gradients of $-\log p(\boldsymbol{x})$ with respect to NADE's parameters.

2.1 Relationship with the RBM

The proposed weight-tying for NADE isn't simply motivated by computational reasons. It also reflects the computations of approximation inference in the RBM.

Denoting the energy function and distribution under an RBM as

$$E(\boldsymbol{x},\boldsymbol{h}) = -\boldsymbol{h}^\top\boldsymbol{W}\boldsymbol{x} - \boldsymbol{b}^\top\boldsymbol{x} - \boldsymbol{c}^\top\boldsymbol{h} \tag{7}$$

$$p(\boldsymbol{x},\boldsymbol{h}) = \exp\left\{-E(\boldsymbol{x},\boldsymbol{h})\right\}/Z, \tag{8}$$

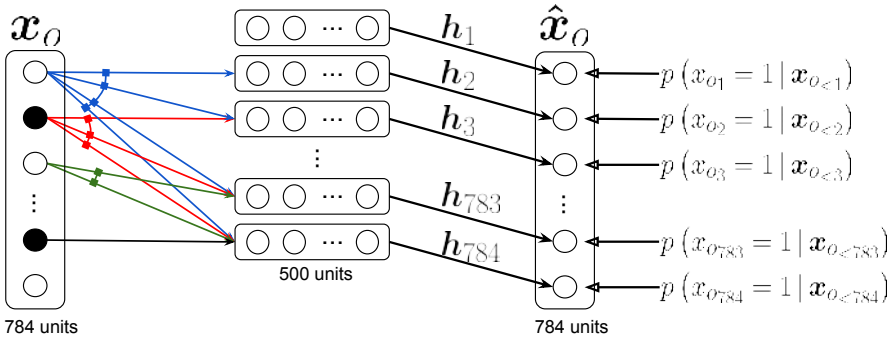


Figure 1: Illustration of a NADE model. In this example, in the input layer, units with value 0 are shown in black while units with value 1 are shown in white. The dashed border represents a layer pre-activation.The outputs $\hat{\boldsymbol{x}}_O$ give predictive probabilities for each dimension of a vector $\boldsymbol{x}_O$, given elements earlier in some ordering. There is no path of connections between an output and the value being predicted, or elements of $\boldsymbol{x}_O$ later in the ordering. Arrows connected together correspond to connections with shared (tied) parameters.

computing all conditionals

$$p(\boldsymbol{x}_{o_d} \mid \boldsymbol{x}_{o_{<d}}) = \sum_{\boldsymbol{x}_{o_{>d}} \in \{0,1\}^{D-d}} \sum_{\boldsymbol{h} \in \{0,1\}^H} \exp\left\{-E(\boldsymbol{x}, \boldsymbol{h})\right\} / Z(\boldsymbol{x}_{o_{<d}}) \tag{9}$$

$$Z(\boldsymbol{x}_{\boldsymbol{o_{<d}}}) = \sum_{\boldsymbol{x}_{o_{\geq d}} \in \{0,1\}^{D-d+1}} \sum_{\boldsymbol{h} \in \{0,1\}^H} \exp\left\{-E(\boldsymbol{x}, \boldsymbol{h})\right\} \tag{10}$$

is intractable. However, these could be approximated using mean-field variational inference. Specifically, consider the conditional over x_{o_d}, $\boldsymbol{x}_{o_{>d}}$ and $\boldsymbol{h}$ instead:

$$p(x_{o_d}, \boldsymbol{x}_{o_{>d}}, \boldsymbol{h} \mid \boldsymbol{x}_{o_{<d}}) = \exp\left\{-E(\boldsymbol{x}, \boldsymbol{h})\right\} / Z(\boldsymbol{x}_{o_{<d}}). \tag{11}$$

A mean-field approach could first approximate this conditional with a factorized distribution

$$q(x_{o_d}, \boldsymbol{x}_{o_{>d}}, \boldsymbol{h} \mid \boldsymbol{x}_{o_{<d}}) = \mu_i(d)^{x_{o_d}} (1 - \mu_d(d))^{1 - x_{o_d}} \prod_{j > d} \mu_j(d)^{x_{o_j}} (1 - \mu_j(d))^{1 - x_{o_j}}$$
$$\prod_k \tau_k(d)^{h_k} (1 - \tau_k(d))^{1 - h_k}, \tag{12}$$

where $\mu_j(d)$ is the marginal probability of x_{o_j} being equal to 1, given $\boldsymbol{x}_{o_{<d}}$. Similarly, $\tau_k(d)$ is the marginal for hidden variable h_k. The dependence on d comes from conditioning on $\boldsymbol{x}_{o_{<d}}$, that is on the first $d-1$ dimensions of $\boldsymbol{x}$ in ordering o.

For some d, a mean-field approximation is obtained by finding the parameters $\mu_j(d)$ for $j \in \{d, \ldots, D\}$ and $\tau_k(d)$ for $k \in \{1, \ldots, H\}$ which minimize the KL divergence between $q(x_{o_d}, \boldsymbol{x}_{o_{>d}}, \boldsymbol{h} \mid \boldsymbol{x}_{o_{<d}})$ and $p(x_{o_d}, \boldsymbol{x}_{o_{>d}}, \boldsymbol{h} \mid \boldsymbol{x}_{o_{<d}})$. This is usually done by finding message

Algorithm 1 Computation of $p(\boldsymbol{x})$ and learning gradients for NADE.

Input: training observation vector $\boldsymbol{x}$ and ordering o of the input dimensions.
Output: $p(\boldsymbol{x})$ and gradients of $-\log p(\boldsymbol{x})$ on parameters.

Computing $p(\boldsymbol{x})$
$\boldsymbol{a}_1 \leftarrow \boldsymbol{c}$
$p(\boldsymbol{x}) \leftarrow 1$
for d from 1 to D **do**
  $\boldsymbol{h}_d \leftarrow \text{sigm}\left(\boldsymbol{a}_d\right)$
  $p(x_{o_d}=1 \mid \boldsymbol{x}_{o_{<d}}) \leftarrow \text{sigm}\left(\boldsymbol{V}_{o_d,\cdot}\boldsymbol{h}_d + b_{o_d}\right)$
  $p(\boldsymbol{x}) \leftarrow p(\boldsymbol{x})\left(p(x_{o_d}=1 \mid \boldsymbol{x}_{o_{<d}})^{x_{o_d}} + (1 - p(x_{o_d}=1 \mid \boldsymbol{x}_{o_{<d}}))^{1-x_{o_d}}\right)$
  $\boldsymbol{a}_{d+1} \leftarrow \boldsymbol{a}_d + \boldsymbol{W}_{\cdot,o_d} x_{o_d}$
end for

Computing gradients of $-\log p(\boldsymbol{x})$
$\delta\boldsymbol{a}_D \leftarrow 0$
$\delta\boldsymbol{c} \leftarrow 0$
for d from D to 1 **do**
  $\delta b_{o_d} \leftarrow \left(p(x_{o_d}=1 \mid \boldsymbol{x}_{o_{<d}}) - x_{o_d}\right)$
  $\delta\boldsymbol{V}_{o_d,\cdot} \leftarrow \left(p(x_{o_d}=1 \mid \boldsymbol{x}_{o_{<d}}) - x_{o_d}\right)\boldsymbol{h}_d^\top$
  $\delta\boldsymbol{h}_d \leftarrow \left(p(x_{o_d}=1 \mid \boldsymbol{x}_{o_{<d}}) - x_{o_d}\right)\boldsymbol{V}_{o_d,\cdot}^\top$
  $\delta\boldsymbol{c} \leftarrow \delta\boldsymbol{c} + \delta\boldsymbol{h}_d \odot \boldsymbol{h}_d \odot (1 - \boldsymbol{h}_d)$
  $\delta\boldsymbol{W}_{\cdot,o_d} \leftarrow \delta\boldsymbol{a}_d x_{o_d}$
  $\delta\boldsymbol{a}_{d-1} \leftarrow \delta\boldsymbol{a}_d + \delta\boldsymbol{h}_d \odot \boldsymbol{h}_d \odot (1 - \boldsymbol{h}_d)$
end for
return $p(\boldsymbol{x}), \delta\boldsymbol{b}, \delta\boldsymbol{V}, \delta\boldsymbol{c}, \delta\boldsymbol{W}$

passing updates that each set the derivatives of the KL divergence to 0 for some of the parameters of $q(x_{o_d}, \boldsymbol{x}_{o_{>d}}, \boldsymbol{h} \mid \boldsymbol{x}_{o_{<d}})$ given others.

For some d, let us fix $\mu_j(d) = x_{o_d}$ for $j < d$, leaving only $\mu_j(d)$ for $j > d$ to be found. The KL-divergence develops as follows:

$$
\begin{aligned}
& \text{KL}(q(x_{o_d}, \boldsymbol{x}_{o_{>d}}, \boldsymbol{h} \mid \boldsymbol{x}_{o_{<d}}) \mid\mid p(x_{o_d}, \boldsymbol{x}_{o_{>d}}, \boldsymbol{h} \mid \boldsymbol{x}_{o_{<d}})) \\
= & -\sum_{x_{o_d}, \boldsymbol{x}_{o_{>d}}, \boldsymbol{h}} q(x_{o_d}, \boldsymbol{x}_{o_{>d}}, \boldsymbol{h} \mid \boldsymbol{x}_{o_{<d}}) \log p(x_{o_d}, \boldsymbol{x}_{o_{>d}}, \boldsymbol{h} \mid \boldsymbol{x}_{o_{<d}}) \\
& + \sum_{x_{o_d}, \boldsymbol{x}_{o_{>d}}, \boldsymbol{h}} q(x_{o_d}, \boldsymbol{x}_{o_{>d}}, \boldsymbol{h} \mid \boldsymbol{x}_{o_{<d}}) \log q(x_{o_d}, \boldsymbol{x}_{o_{>d}}, \boldsymbol{h} \mid \boldsymbol{x}_{o_{<d}}) \\
= & \log Z(\boldsymbol{x}_{o_{<d}}) - \sum_j \sum_k \tau_k(d) W_{k,o_j} \mu_j(d) - \sum_j b_{o_j} \mu_j(d) - \sum_k c_k \tau_k(d) \\
& + \sum_{j \geq d} \left(\mu_j(d) \log \mu_j(d) + (1 - \mu_j(d)) \log(1 - \mu_j(d))\right) \\
& + \sum_k \left(\tau_k(d) \log \tau_k(d) + (1 - \tau_k(d)) \log(1 - \tau_k(d))\right).
\end{aligned}
$$

Then, we can take the derivative with respect to $\tau_k(d)$ and set it to 0, to obtain:

$$\begin{aligned}
0 &= \frac{\partial \mathrm{KL}(q(x_{o_d}, \boldsymbol{x}_{o_{>d}}, \boldsymbol{h} \,|\, \boldsymbol{x}_{o_{<d}}) \,||\, p(x_{o_d}, \boldsymbol{x}_{o_{>d}}, \boldsymbol{h} \,|\, \boldsymbol{x}_{o_{<d}}))}{\partial \tau_k(d)} \\
0 &= -c_k - \sum_j W_{k,o_j} \mu_j(d) + \log\left(\frac{\tau_k(d)}{1-\tau_k(d)}\right) \\
\frac{\tau_k(d)}{1-\tau_k(d)} &= \exp\left\{c_k + \sum_j W_{k,o_j} \mu_j(d)\right\} \qquad (13) \\
\tau_k(d) &= \frac{\exp\left\{c_k + \sum_j W_{k,o_j} \mu_j(d)\right\}}{1 + \exp\left\{c_k + \sum_j W_{k,o_j} \mu_j(d)\right\}} \\
\tau_k(d) &= \mathrm{sigm}\left(c_k + \sum_{j \geq d} W_{k,o_j} \mu_j(d) + \sum_{j<d} W_{k,o_j} x_{o_j}\right). \qquad (14)
\end{aligned}$$

where in the last step we have used the fact that $\mu_j(d) = x_{o_j}$ for $j < d$. Equation 14 would correspond to the message passing updates of the hidden unit marginals $\tau_k(d)$ given the marginals of input $\mu_j(d)$.

Similarly, we can set the derivative with respect to $\mu_j(d)$ for $j \geq d$ to 0 and obtain:

$$\begin{aligned}
0 &= \frac{\partial \mathrm{KL}(q(x_{o_d}, \boldsymbol{x}_{o_{>d}}, \boldsymbol{h} \,|\, \boldsymbol{x}_{o_{<d}}) \,||\, p(x_{o_d}, \boldsymbol{x}_{o_{>d}}, \boldsymbol{h} \,|\, \boldsymbol{x}_{o_{<d}}))}{\partial \mu_j(d)} \\
0 &= -b_{o_d} - \sum_k \tau_k(d) W_{k,o_j} + \log\left(\frac{\mu_j(d)}{1-\mu_j(d)}\right) \\
\frac{\mu_j(d)}{1-\mu_j(d)} &= \exp\left\{b_{o_j} + \sum_k \tau_k(d) W_{k,o_j}\right\} \\
\mu_j(d) &= \frac{\exp\left\{b_{o_j} + \sum_k \tau_k(d) W_{k,o_j}\right\}}{1 + \exp\left\{b_{o_j} + \sum_k \tau_k(d) W_{k,o_j}\right\}} \\
\mu_j(d) &= \mathrm{sigm}\left(b_{o_j} + \sum_k \tau_k(d) W_{k,o_j}\right). \qquad (15)
\end{aligned}$$

Equation 15 would correspond to the message passing updates of the input marginals $\mu_j(d)$ given the hidden layer marginals $\tau_k(d)$. The complete mean-field algorithm would thus alternate between applying the updates of Equations 14 and 15, right to left.

We now notice that Equation 14 corresponds to NADE's hidden layer computation (Equation 3) where $\mu_j(d) = 0 \;\; \forall j \geq d$. Also, Equation 15 corresponds to NADE's output layer computation (Equation 2) where $j = d$, $\tau_k(d) = h_{d,k}$ and $\boldsymbol{W}^\top = \boldsymbol{V}$. Thus, in short, NADE's forward pass is equivalent to applying a single pass of mean-field inference to approximate all the conditionals $p(\boldsymbol{x}_{o_d} \,|\, \boldsymbol{x}_{o_{<d}})$ of an RBM, where initially $\mu_j(d) = 0$ and where a separate matrix $\boldsymbol{V}$ is used for the hidden-to-input messages. A generalization of NADE based on this connection to mean field inference has been further explored by Raiko et al. (2014).

3. NADE for Non-Binary Observations

So far we have only considered the case of binary observations x_i. However, the framework of NADE naturally extends to distributions over other types of observations.

In the next section, we discuss the case of real-valued observations, which is one of the most general cases of non-binary observations and provides an illustrative example of the technical considerations one faces when extending NADE to new observations.

3.1 RNADE: Real-Valued NADE

A NADE model for real-valued data could be obtained by applying the derivations shown in Section 2.1 to the Gaussian-RBM (Welling et al., 2005). The resulting neural network would output the mean of a Gaussian with fixed variance for each of the conditionals in Equation 1. Such a model is not competitive with mixture models, for example on perceptual data sets (Uria, 2015). However, we can explore alternative models by making the neural network for each conditional distribution output the parameters of a distribution that's not a fixed-variance Gaussian.

In particular, a mixture of one-dimensional Gaussians for each autoregressive conditional provides a flexible model. Given enough components, a mixture of Gaussians can model any continuous distribution to arbitrary precision. The resulting model can be interpreted as a sequence of mixture density networks (Bishop, 1994) with shared parameters. We call this model RNADE-MoG. In RNADE-MoG, each of the conditionals is modeled by a mixture of Gaussians:

$$p(x_{o_d} \,|\, \boldsymbol{x}_{o_{<d}}) = \sum_{c=1}^{C} \pi_{o_d,c}\, \mathcal{N}(x_{o_d};\, \mu_{o_d,c},\, \sigma^2_{o_d,c}), \tag{16}$$

where the parameters are set by the outputs of a neural network:

$$\pi_{o_d,c} = \frac{\exp\left\{z^{(\pi)}_{o_d,c}\right\}}{\sum_{c=1}^{C} \exp\left\{z^{(\pi)}_{o_d,c}\right\}} \tag{17}$$

$$\mu_{o_d,c} = z^{(\mu)}_{o_d,c} \tag{18}$$

$$\sigma_{o_d,c} = \exp\left\{z^{(\sigma)}_{o_d,c}\right\} \tag{19}$$

$$z^{(\pi)}_{o_d,c} = b^{(\pi)}_{o_d,c} + \sum_{k=1}^{H} V^{(\pi)}_{o_d,k,c} h_{d,k} \tag{20}$$

$$z^{(\mu)}_{o_d,c} = b^{(\mu)}_{o_d,c} + \sum_{k=1}^{H} V^{(\mu)}_{o_d,k,c} h_{d,k} \tag{21}$$

$$z^{(\sigma)}_{o_d,c} = b^{(\sigma)}_{o_d,c} + \sum_{k=1}^{H} V^{(\sigma)}_{o_d,k,c} h_{d,k} \tag{22}$$

Parameter sharing conveys the same computational and statistical advantages as it does in the binary NADE.

Different one dimensional conditional forms may be preferred, for example due to limited data set size or domain knowledge about the form of the conditional distributions. Other choices, like single variable-variance Gaussians, sinh-arcsinh distributions, and mixtures of Laplace distributions, have been examined by Uria (2015).

Training an RNADE can still be done by stochastic gradient descent on the parameters of the model with respect to the negative log-density of the training set. It was found empirically (Uria et al., 2013) that stochastic gradient descent leads to better parameter configurations when the gradient of the mean $\left(\frac{\partial J}{\partial \mu_{o_d,c}}\right)$ was multiplied by the standard deviation ($\sigma_{o_d,c}$).

4. Orderless and Deep NADE

The fixed ordering of the variables in a NADE model makes the exact calculation of arbitrary conditional probabilities computationally intractable. Only a small subset of conditional distributions, those where the conditioned variables are at the beginning of the ordering and marginalized variables at the end, are computationally tractable.

Another limitation of NADE is that a naive extension to a deep version, with multiple layers of hidden units, is computationally expensive. Deep neural networks (Bengio, 2009; LeCun et al., 2015) are at the core of state-of-the-art models for supervised tasks like image recognition (Krizhevsky et al., 2012) and speech recognition (Dahl et al., 2013). The same inductive bias should also provide better unsupervised models. However, extending the NADE framework to network architectures with several hidden layers, by introducing extra non-linear calculations between Equations 3 and 2, increases its complexity to cubic in the number of units per layer. Specifically, the cost becomes $O(DH^2L)$, where L stands for the number of hidden layers and can be assumed to be a small constant, D is the number of variables modeled, and H is the number of hidden units, which we assumed to be of the same order as D. This increase in complexity is caused by no longer being able to share hidden layer computations across the conditionals in Equation 1, after the non-linearity in the first layer.

In this section we describe an order-agnostic training procedure, DeepNADE (Uria et al., 2014), which will address both of the issues above. This procedure trains a single deep neural network that can assign a conditional distribution to any variable given any subset of the others. This network can then provide the conditionals in Equation 1 for any ordering of the input observations. Therefore, the network defines a factorial number of different models with shared parameters, one for each of the $D!$ orderings of the inputs. At test time, given an inference task, the most convenient ordering of variables can be used. The models for different orderings will not be consistent with each other: they will assign different probabilities to a given test vector. However, we can use the models' differences to our advantage by creating ensembles of NADE models (Section 4.1), which results in better estimators than any single NADE. Moreover, the training complexity of our procedure increases linearly with the number of hidden layers $O(H^2L)$, while remaining quadratic in the size of the network's layers.

We first describe the model for an L-layer neural network modeling binary variables. A conditional distribution is obtained directly from a hidden unit in the final layer:

$$p(x_{o_d}=1 \,|\, \boldsymbol{x}_{o_{<d}}, \boldsymbol{\theta}, o_{<d}, o_d) = \boldsymbol{h}^{(L)}_{o_d}. \tag{23}$$

This hidden unit is computed from previous layers, all of which can only depend on the $\boldsymbol{x}_{o_{<d}}$ variables that are currently being conditioned on. We remove the other variables from the computation using a binary mask,

$$\boldsymbol{m}_{o_{<d}} = [1_{1\in o_{<d}}, 1_{2\in o_{<d}}, \ldots, 1_{D\in o_{<d}}], \tag{24}$$

which is element-wise multiplied with the inputs before computing the remaining layers as in a standard neural network:

$$\boldsymbol{h}^{(0)} = \boldsymbol{x} \odot \boldsymbol{m}_{o_{<d}} \tag{25}$$

$$\boldsymbol{a}^{(\ell)} = \boldsymbol{W}^{(\ell)}\boldsymbol{h}^{(\ell-1)} + \boldsymbol{b}^{(\ell)} \tag{26}$$

$$\boldsymbol{h}^{(\ell)} = \boldsymbol{\sigma}\left(\boldsymbol{a}^{(\ell)}\right) \tag{27}$$

$$\boldsymbol{h}^{(L)} = \mathbf{sigm}\left(\boldsymbol{a}^{(L)}\right). \tag{28}$$

The network is specified by a free choice of the activation function $\boldsymbol{\sigma}(\cdot)$, and learnable parameters $\boldsymbol{W}^{(\ell)} \in \mathbb{R}^{H^{(\ell)}\times H^{(\ell-1)}}$ and $\boldsymbol{b}^{(\ell)} \in \mathbb{R}^{H^{(\ell)}}$, where $H^{(l)}$ is the number of units in the ℓ-th layer. As layer zero is the masked input, $H^{(0)} = D$. The final L-th layer needs to be able to provide predictions for any element (Equation 23) and so also has D units.

To train a DeepNADE, the ordering of the variables is treated as a stochastic variable with a uniform distribution. Moreover, since we wish DeepNADE to provide good predictions for any ordering, we optimize the expected likelihood over the ordering of variables:

$$\mathcal{J}(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{o\in D!} -\log p(\boldsymbol{X} \,|\, \boldsymbol{\theta}, o) \propto \mathop{\mathbb{E}}_{o\in D!} \mathop{\mathbb{E}}_{\boldsymbol{x}\in\mathcal{X}} -\log p(\boldsymbol{x} \,|\, \boldsymbol{\theta}, o), \tag{29}$$

where we've made the dependence on the ordering o and the network's parameters $\boldsymbol{\theta}$ explicit, $D!$ stands for the set of all orderings (the permutations of D elements) and $\boldsymbol{x}$ is a uniformly sampled data point from the training set $\mathcal{X}$. Using NADE's expression for the density of a data point in Equation 1 we have

$$\mathcal{J}(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{o\in D!} \mathop{\mathbb{E}}_{\boldsymbol{x}\in\mathcal{X}} \sum_{d=1}^{D} -\log p(x_{o_d} \,|\, \boldsymbol{x}_{o_{<d}}, \boldsymbol{\theta}, o), \tag{30}$$

where d indexes the elements in the ordering, o, of the variables. By moving the expectation over orderings inside the sum over the elements of the ordering, the ordering can be split in three parts: $o_{<d}$ (the indices of the $d-1$ first dimensions in the ordering), o_d (the index of the d-th variable) and $o_{>d}$ (the indices of the remaining dimensions). Therefore, the loss function can be rewritten as:

$$\mathcal{J}(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\boldsymbol{x}\in\mathcal{X}} \sum_{d=1}^{D} \mathop{\mathbb{E}}_{o_{<d}} \mathop{\mathbb{E}}_{o_d} \mathop{\mathbb{E}}_{o_{>d}} -\log p(x_{o_d} \,|\, \boldsymbol{x}_{o_{<d}}, \boldsymbol{\theta}, o_{<d}, o_d, o_{>d}). \tag{31}$$

The value of each of these terms does not depend on $o_{>d}$. Therefore, it can be simplified as:

$$\mathcal{J}(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\boldsymbol{x}\in\mathcal{X}} \sum_{d=1}^{D} \mathop{\mathbb{E}}_{o_{<d}} \mathop{\mathbb{E}}_{o_d} -\log p(x_{o_d} \,|\, \boldsymbol{x}_{o_{<d}}, \boldsymbol{\theta}, o_{<d}, o_d). \tag{32}$$

In practice, this loss function will have a very high number of terms and will have to be approximated by sampling $\boldsymbol{x}$, d and $o_{<d}$. The innermost expectation over values of o_d can be calculated cheaply, because all of the neural network computations depend only on the masked input $\boldsymbol{x}_{o_{<d}}$, and can be reused for each possible o_d. Assuming all orderings are equally probable, we will estimate $\mathcal{J}(\boldsymbol{\theta})$ by:

$$\widehat{\mathcal{J}}(\boldsymbol{\theta}) = \frac{D}{D-d+1} \sum_{o_d} -\log p(x_{o_d} \,|\, \boldsymbol{x}_{o_{<d}}, \boldsymbol{\theta}, o_{<d}, o_d), \tag{33}$$

which is an unbiased estimator of Equation 29. Therefore, training can be done by descent on the gradient of $\widehat{\mathcal{J}}(\boldsymbol{\theta})$.

For binary observations, we use the cross-entropy scaled by a factor of $\frac{D}{D-d+1}$ as the training loss which corresponds to minimizing $\widehat{\mathcal{J}}$:

$$\mathcal{J}(\boldsymbol{x}) = \frac{D}{D-d+1} \boldsymbol{m}_{o_{\geq d}}^{\top} \left(\boldsymbol{x} \odot \mathbf{log}\left(\boldsymbol{h}^{(L)}\right) + (1-\boldsymbol{x}) \odot \mathbf{log}\left(1-\boldsymbol{h}^{(L)}\right) \right). \tag{34}$$

Differentiating this cost involves backpropagating the gradients of the cross-entropy only from the outputs in $o_{\geq d}$ and rescaling them by $\frac{D}{D-d+1}$.

The resulting training procedure resembles that of a denoising autoencoder (Vincent et al., 2008). Like the autoencoder, D outputs are used to predict D inputs corrupted by a random masking process ($\boldsymbol{m}_{o_{<d}}$ in Equation 25). A single forward pass can compute $\boldsymbol{h}^{(L)}_{o_{\geq d}}$, which provides a prediction $p(x_{o_d} = 1 \,|\, \boldsymbol{x}_{o_{<d}}, \boldsymbol{\theta}, o_{<d}, o_d)$ for every masked variable, which could be used next in an ordering starting with $o_{<d}$. Unlike the autoencoder, the outputs for variables corresponding to those provided in the input (not masked out) are ignored.

In this order-agnostic framework, missing variables and zero-valued observations are indistinguishable by the network. This shortcoming can be alleviated by concatenating the inputs to the network (masked variables $\boldsymbol{x} \odot \boldsymbol{m}_{o_{<d}}$) with the mask $\boldsymbol{m}_{o_{<d}}$. Therefore we advise substituting the input described in Equation 25 with

$$\boldsymbol{h}^{(0)} = \text{concat}(\boldsymbol{x} \odot \boldsymbol{m}_{o_{<d}}, \boldsymbol{m}_{o_{<d}}). \tag{35}$$

We found this modification to be important in order to obtain competitive statistical performance (see Table 3). The resulting neural network is illustrated in Figure 2.

4.1 Ensembles of NADE Models

As mentioned, the DeepNADE parameter fitting procedure effectively produces a factorial number of different NADE models, one for each ordering of the variables. These models will not, in general, assign the same probability to any particular data point. This disagreement is undesirable if we require consistent inferences for different inference problems, as it will preclude the use of the most convenient ordering of variables for each inference task.

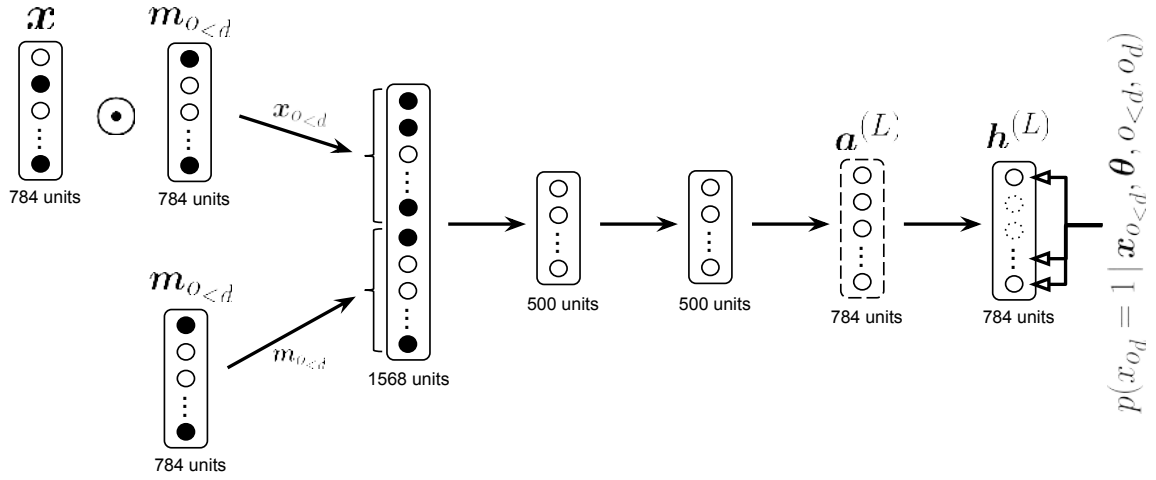


Figure 2: Illustration of a DeepNADE model with two hidden layers. The dashed border represents a layer pre-activation. Units with value 0 are shown in black while units with value 1 are shown in white. A mask $\boldsymbol{m}_{o_{<d}}$ specifies a subset of variables to condition on. A conditional or predictive probability of the remaining variables is given in the final layer. The output units with a corresponding input mask of value 1 (shown with dotted contour) are not involved in DeepNADE's training loss (Equation 34).

However, it is possible to use this variability across the different orderings to our advantage by combining several models. A usual approach to improve on a particular estimator is to construct an ensemble of multiple, strong but different estimators, e.g. using bagging (Ormoneit and Tresp, 1995) or stacking (Smyth and Wolpert, 1999). The DeepNADE training procedure suggests a way of generating ensembles of NADE models: take a set of uniformly distributed orderings $\{o^{(k)}\}_{k=1}^{K}$ over the input variables and use the average probability $\frac{1}{K}\sum_{k=1}^{K} p(\boldsymbol{x} \mid \theta, o^{(k)})$ as an estimator.

The use of an ensemble increases the test-time cost of density estimation linearly with the number of orderings used. The complexity of sampling does not change however: after one of the K orderings is chosen at random, the single corresponding NADE is sampled. Importantly, the cost of training also remains the same, unlike other ensemble methods such as bagging. Furthermore, the number of components can be chosen after training and even adapted to a computational budget on the fly.

5. ConvNADE: Convolutional NADE

One drawback of NADE (and its variants so far) is the lack of a mechanism for truly exploiting the high-dimensional structure of the data. For example, when using NADE on binarized MNIST, we first need to flatten the 2D images before providing them to the model as a vector. As the spatial topology is not provided to the network, it can't use this information to share parameters and may learn less quickly.

Recently, convolutional neural networks (CNN) have achieved state-of-the-art performance on many supervised tasks related to images Krizhevsky et al. (2012). Briefly, CNNs are composed of convolutional layers, each one having multiple learnable filters. The outputs of a convolutional layer are feature maps and are obtained by the convolution on the input image (or previous feature maps) of a linear filter, followed by the addition of a bias and the application of a non-linear activation function. Thanks to the convolution, spatial structure in the input is preserved and can be exploited. Moreover, as per the definition of a convolution the same filter is reused across all sub-regions of the entire image (or previous feature maps), yielding a parameter sharing that is natural and sensible for images.

The success of CNNs raises the question: can we exploit the spatial topology of the inputs while keeping NADE's autoregressive property? It turns out we can, simply by replacing the fully connected hidden layers of a DeepNADE model with convolutional layers. We thus refer to this variant as Convolutional NADE (ConvNADE).

First we establish some notation that we will use throughout this section. Without loss of generality, let the input $\boldsymbol{X} \in \{0,1\}^{N_X \times N_X}$ be a square binary image of size N_X and every convolution filter $\boldsymbol{W}_{ij}^{(\ell)} \in \mathbb{R}^{N_W^{(\ell)} \times N_W^{(\ell)}}$ connecting two feature maps $\boldsymbol{H}_i^{(\ell-1)}$ and $\boldsymbol{H}_j^{(\ell)}$ also be square with their size $N_W^{(\ell)}$ varying for each layer ℓ. We also define the following mask $\boldsymbol{M}_{o_{<d}} \in \{0,1\}^{N_X \times N_X}$, which is 1 for the locations of the first $d-1$ pixels in the ordering o.

Formally, Equation 26 is modified to use convolutions instead of dot products. Specifically for an L-layer convolutional neural network that preserves the input shape (explained below) we have

$$p(x_{o_d} = 1 \,|\, \boldsymbol{x}_{o_{<d}}, \boldsymbol{\theta}, o_{<d}, o_d) = \mathbf{vec}\left(\boldsymbol{H}_1^{(L)}\right)_{o_d}, \tag{36}$$

with

$$\boldsymbol{H}_1^{(0)} = \boldsymbol{X} \odot \boldsymbol{M}_{o_{<d}} \tag{37}$$

$$\boldsymbol{A}_j^{(\ell)} = b_j^{(\ell)} + \sum_{i=1}^{H^{(\ell-1)}} \boldsymbol{H}_i^{(\ell-1)} \circledast \boldsymbol{W}_{ij}^{(\ell)} \tag{38}$$

$$\boldsymbol{H}_j^{(\ell)} = \boldsymbol{\sigma}\left(\boldsymbol{A}_j^{(\ell)}\right) \tag{39}$$

$$\boldsymbol{H}_j^{(L)} = \mathbf{sigm}\left(\boldsymbol{A}_j^{(L)}\right), \tag{40}$$

where $H^{(\ell)}$ is the number of feature maps output by the ℓ-th layer and $\boldsymbol{b}^{(l)} \in \mathbb{R}^{H^{(l)}}$, $\boldsymbol{W}^{(\ell)} \in \mathbb{R}^{H^{(\ell-1)} \times H^{(\ell)} \times N_W^{(\ell)} \times N_W^{(\ell)}}$, with $\odot$ denoting the element-wise multiplication, $\boldsymbol{\sigma}(\cdot)$ being any activation function and $\mathbf{vec}(\boldsymbol{X}) \to \boldsymbol{x}$ is the concatenation of every row in $\boldsymbol{X}$. Note that $H^{(0)}$ corresponds to the number of channels the input images have.

For notational convenience, we use $\circledast$ to denote both "valid" convolutions and "full" convolutions, instead of introducing bulky notations to differentiate these cases. The "valid" convolutions only apply a filter to complete patches of the image, resulting in a smaller image (its shape is decreased to $N_X - N_W^{(\ell)} + 1$). Alternatively, "full" convolutions zero-pad the contour of the image before applying the convolution, thus expanding the image (its shape is increased to $N_X + N_W^{(\ell)} - 1$). Which one is used should be self-explanatory depending on the context. Note that we only use convolutions with a stride of 1.

Moreover, in order for ConvNADE to output conditional probabilities as shown in Equation 36, the output layer must have only one feature map $\boldsymbol{H}_1^{(L)}$, whose dimension matches the dimension of the input $\boldsymbol{X}$. This can be achieved by carefully combining layers that use either "valid" or "full" convolutions.

To explore different model architectures respecting that constraint, we opted for the following strategy. Given a network, we ensured the first half of its layers was using "valid" convolutions while the other half would use "full" convolutions. In addition to that, we made sure the network was symmetric with respect to its filter shapes (i.e. the filter shape used in layer ℓ matched the one used in layer $L - \ell$).

For completeness, we wish to mention that ConvNADE can also include pooling and upsampling layers, but we did not see much improvement when using them. In fact, recent research suggests that these types of layers are not essential to obtain state-of-the-art results (Springenberg et al., 2015).

The flexibility of DeepNADE allows us to easily combine both convolutional and fully connected layers. To create such hybrid models, we used the simple strategy of having two separate networks, with their last layer fused together at the end. The 'convnet' part is only composed of convolutional layers whereas the 'fullnet' part is only composed of fully connected layers. The forward pass of both networks follows respectively Equations 37–39 and Equations 25–27. Note that in the 'fullnet' network case, $\boldsymbol{x}$ corresponds to the input image having been flattened.

In the end, the output layer $\boldsymbol{g}$ of the hybrid model corresponds to the aggregation of the last layer pre-activation of both 'convnet' and 'fullnet' networks. The conditionals are slightly modified as follows:

$$p(x_{o_d} = 1 \mid \boldsymbol{x}_{o_{<d}}, \boldsymbol{\theta}, o_{<d}, o_d) = \boldsymbol{g}_{o_d} \tag{41}$$

$$\boldsymbol{g} = \mathbf{sigm}\left(\mathbf{vec}\left(\boldsymbol{A}_1^{(L)}\right) + \boldsymbol{a}^{(L)}\right). \tag{42}$$

The same training procedure as for DeepNADE model can also be used for ConvNADE. For binary observations, the training loss is similar to Equation 34, with $\boldsymbol{h}^{(L)}$ being substituted for $\boldsymbol{g}$ as defined in Equation 42.

As for the DeepNADE model, we found that providing the mask $\boldsymbol{M}_{o_{<d}}$ as an input to the model improves performance (see Table 4). For the 'convnet' part, the mask was provided as an additional channel to the input layer. For the 'fullnet' part, the inputs were concatenated with the mask as shown in Equation 35.

The final architecture is shown in Figure 3. In our experiments, we found that this type of hybrid model works better than only using convolutional layers (see Table 4). Certainly, more complex architectures could be employed but this is a topic left for future work.

6. Related Work

As we mentioned earlier, the development of NADE and its extensions was motivated by the question of whether a tractable distribution estimator could be designed to match a powerful but intractable model such as the restricted Boltzmann machine.

The original inspiration came from the autoregressive approach taken by fully visible sigmoid belief networks (FVSBN), which were shown by Frey et al. (1996) to be surprisingly

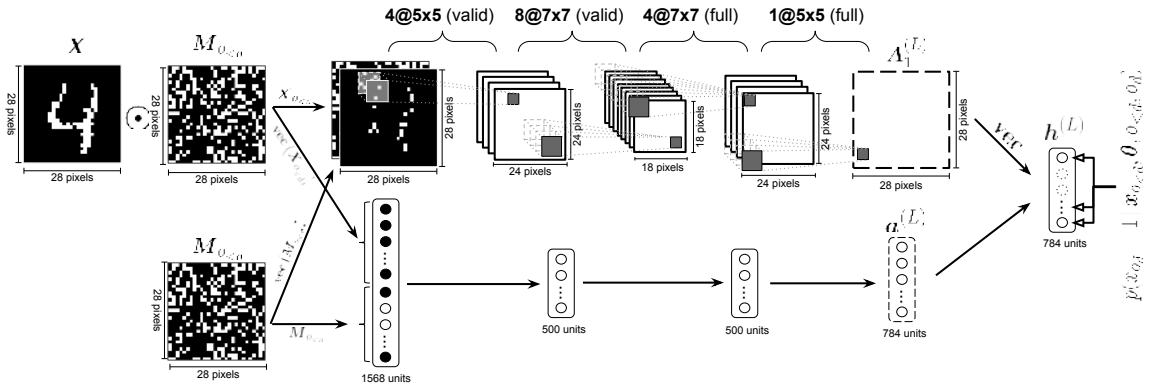


Figure 3: Illustration of a ConvNADE that combines a convolutional neural network with three hidden layers and a fully connected feed-forward neural network with two hidden layers. The dashed border represents a layer pre-activation. Units with a dotted contour are not valid conditionals since they depend on themselves i.e. they were given in the input.

competitive, despite the simplicity of the distribution family for its conditionals. Bengio and Bengio (2000) later proposed using more powerful conditionals, modeled as single layer neural networks. Moreover, they proposed connecting the output of each d^{th} conditional to all of the hidden layers of the $d-1$ neural networks for the preceding conditionals. More recently, Germain et al. (2015) generalized this model by deriving a simple procedure for making it deep and orderless (akin to DeepNADE, in Section 4). We compare with all of these approaches in Section 7.1.

There exists, of course, more classical and non-autoregressive approaches to tractable distribution estimation, such as mixture models and Chow–Liu trees (Chow and Liu, 1968). We compare with these as well in Section 7.1.

This work also relates directly to the recently growing literature on generative neural networks. In addition to the autoregressive approach described in this paper, there exists three other types of such models: directed generative networks, undirected generative networks and hybrid networks.

Work on directed generative networks dates back to the original work on sigmoid belief networks (Neal, 1992) and the Helmholtz machine (Hinton et al., 1995; Dayan et al., 1995). Helmholtz machines are equivalent to a multilayer sigmoid belief network, with each using binary stochastic units. Originally they were trained using Gibbs sampling and gradient descent (Neal, 1992), or with the so-called wake sleep algorithm (Hinton et al., 1995). More recently, many alternative directed models and training procedures have been proposed. Kingma and Welling (2014); Rezende et al. (2014) proposed the variational autoencoder (VAE), where the model is the same as the Helmholtz machine, but with real-valued (usually Gaussian) stochastic units. Importantly, Kingma and Welling (2014) identified a reparameterization trick making it possible to train the VAE in a way that resembles the training of an autoencoder. This approach falls in the family of stochastic variational inference methods, where the encoder network corresponds to the approximate variational

posterior. The VAE optimizes a bound on the likelihood which is estimated using a single sample from the variational posterior, though recent work has shown that a better bound can be obtained using an importance sampling approach (Burda et al., 2016). Gregor et al. (2015) later exploited the VAE approach to develop DRAW, a directed generative model for images based on a read-write attentional mechanism. Goodfellow et al. (2014) proposed an adversarial approach to training directed generative networks, that relies on a discriminator network simultaneously trained to distinguish between data and model samples. Generative networks trained this way are referred to as Generative Adversarial Networks (GAN). While the VAE optimizes a bound of the likelihood (which is the KL divergence between the empirical and model distributions), it can be shown that the earliest versions of GANs optimize the Jensen–Shannon (JS) divergence between the empirical and model distributions. Li et al. (2015) instead propose a training objective derived from Maximum Mean Discrepancy (MMD; Gretton et al., 2007). Recently, the directed generative model approach has been very successfully applied to model images (Denton et al., 2015; Sohl-Dickstein et al., 2011).

The undirected paradigm has also been explored extensively for developing powerful generative networks. These include the restricted Boltzmann machine (Smolensky, 1986; Freund and Haussler, 1992) and its multilayer extension, the deep Boltzmann machine (Salakhutdinov and Hinton, 2009), which dominate the literature on undirected neural networks. Salakhutdinov and Murray (2008) provided one of the first quantitative evidence of the generative modeling power of RBMs, which motivated the original parameterization for NADE (Larochelle and Murray, 2011). Efforts to train better undirected models can vary in nature. One has been to develop alternative objectives to maximum likelihood. The proposal of Contrastive Divergence (CD; Hinton, 2002) was instrumental in the popularization of the RBM. Other proposals include pseudo-likelihood (Besag, 1975; Marlin et al., 2010), score matching (Hyvärinen, 2005; Hyvärinen, 2007a,b), noise contrastive estimation (Gutmann and Hyvärinen, 2010) and probability flow minimization (Sohl-Dickstein et al., 2011). Another line of development has been to optimize likelihood using Robbins–Monro stochastic approximation (Younes, 1989), also known as Persistent CD (Tieleman, 2008), and develop good MCMC samplers for deep undirected models (Salakhutdinov, 2009, 2010; Desjardins et al., 2010; Cho et al., 2010). Work has also been directed towards proposing improved update rules or parameterization of the model's energy function (Tieleman and Hinton, 2009; Cho et al., 2013; Montavon and Müller, 2012) as well as improved approximate inference of the hidden layers (Salakhutdinov and Larochelle, 2010). The work of Ngiam et al. (2011) also proposed an undirected model that distinguishes itself from deep Boltzmann machines by having deterministic hidden units, instead of stochastic.

Finally, hybrids of directed and undirected networks are also possible, though much less common. The most notable case is the Deep Belief Network (DBN; Hinton et al., 2006), which corresponds to a sigmoid belief network for which the prior over its top hidden layer is an RBM (whose hidden layer counts as an additional hidden layer). The DBN revived interest in RBMs, as they were required to successfully initialize the DBN.

NADE thus substantially differs from this literature focusing on directed and undirected models, benefiting from a few properties that these approaches lack. Mainly, NADE does not rely on latent stochastic hidden units, making it possible to tractably compute its associated data likelihood for some given ordering. This in turn makes it possible to efficiently produce

Name	# Inputs	Train	Valid.	Test
Adult	123	5000	1414	26147
Connect4	126	16000	4000	47557
DNA	180	1400	600	1186
Mushrooms	112	2000	500	5624
NIPS-0-12	500	400	100	1240
OCR-letters	128	32152	10000	10000
RCV1	150	40000	10000	150000
Web	300	14000	3188	32561

Table 1: Statistics on the binary vector data sets of Section 7.1.

exact samples from the model (unlike in undirected models) and get an unbiased gradient for maximum likelihood training (unlike in directed graphical models).

7. Results

In this section, we evaluate the performance of our different NADE models on a variety of data sets. The code to reproduce the experiments of the paper is available on GitHub[1]. Our implementation is done using Theano (Team et al., 2016).

7.1 Binary Vectors Data Sets

We start by evaluating the performance of NADE models on a set of benchmark data sets where the observations correspond to binary vectors. These data sets were mostly taken from the LIBSVM data sets web site[2], except for OCR-letters[3] and NIPS-0-12 [4]. Code to download these data sets is available here: `http://info.usherbrooke.ca/hlarochelle/code/nade.tar.gz`. Table 1 summarizes the main statistics for these data sets.

For these experiments, we only consider tractable distribution estimators, where we can evaluate $p(\boldsymbol{x})$ on test items exactly. We consider the following baselines:

- **MoB**: A mixture of multivariate Bernoullis, trained using the EM algorithm. The number of mixture components was chosen from $\{32, 64, 128, 256, 512, 1024\}$ based on validation set performance, and early stopping was used to determine the number of EM iterations.

- **RBM**: A restricted Boltzmann machine made tractable by using only 23 hidden units, trained by contrastive divergence with up to 25 steps of Gibbs sampling. The validation set performance was used to select the learning rate from $\{0.005, 0.0005, 0.00005\}$, and the number of iterations over the training set from $\{100, 500, 1000\}$.

1. `http://github.com/MarcCote/NADE`
2. `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`
3. `http://ai.stanford.edu/~btaskar/ocr/`
4. `http://www.cs.nyu.edu/~roweis/data.html`

- **FVSBN**: Fully visible sigmoid belief network, that models each conditional $p(x_{o_d} \mid \boldsymbol{x}_{o_{<d}})$ with logistic regression. The ordering of inputs was selected randomly. Training was by stochastic gradient descent. The validation set was used for early stopping, as well as for choosing the base learning rate $\eta \in \{0.05, 0.005, 0.0005\}$, and a decreasing schedule constant γ from $\{0, 0.001, 0.000001\}$ for the learning rate schedule $\eta/(1+\gamma t)$ for the t^{th} update.

- **Chow–Liu**: A Chow–Liu tree is a graph over the observed variables, where the distribution of each variable, except the root, depends on a single parent node. There is an $O(D^2)$ fitting algorithm to find the maximum likelihood tree and conditional distributions (Chow and Liu, 1968). We adapted an implementation provided by Harmeling and Williams (2011), who found Chow–Liu to be a strong baseline.

  The maximum likelihood parameters are not defined when conditioning on events that haven't occurred in the training set. Moreover, conditional probabilities of zero are possible, which could give infinitely bad test set performance. We re-estimated the conditional probabilities on the Chow–Liu tree using Lidstone or "add-α" smoothing:

$$p(x_d = 1 \mid x_{\text{parent}} = z) = \frac{\text{count}(x_d = 1 \mid x_{\text{parent}} = z) + \alpha}{\text{count}(x_{\text{parent}} = z) + 2\alpha}, \tag{43}$$

  selecting α for each data set from $\{10^{-20}, 0.001, 0.01, 0.1\}$ based on performance on the validation set.

- **MADE** (Germain et al., 2015): Generalization of the neural network approach of Bengio and Bengio (2000), to multiple layers. We consider a version using a single (fixed) input ordering and another trained on multiple orderings from which an ensemble was constructed (which was inspired from the order-agnostic approach of Section 4) that we refer to as MADE-E. See Germain et al. (2015) for more details.

We compare these baselines with the two following NADE variants:

- **NADE (fixed order)**: Single layer NADE model, trained on a single (fixed) randomly generated order, as described in Section 2. The sigmoid activation function was used for the hidden layer, of size 500. Much like for FVSBN, training relied on stochastic gradient descent and the validation set was used for early stopping, as well as for choosing the learning rate from {0.05, 0.005, 0.0005}, and the decreasing schedule constant γ from {0,0.001,0.000001}.

- **NADE-E**: Single layer NADE trained according to the order-agnostic procedure described in Section 4. The rectified linear activation function was used for the hidden layer, also of size 500. Minibatch gradient descent was used for training, with minibatches of size 100. The initial learning rate, chosen among {0.016, 0.004, 0.001, 0.00025, 0.0000675}, was linearly decayed to zero over the course of 100,000 parameter updates. Early stopping was used, using Equation 34 to get a stochastic estimate of the validation set average log-likelihood. An ensemble using 16 orderings was used to compute the test-time log-likelihood.

Model	Adult	Connect4	DNA	Mushrooms	NIPS-0-12	OCR-letters	RCV1	Web
MoB	-20.44	-23.41	-98.19	-14.46	-290.02	-40.56	-47.59	-30.16
RBM	-16.26	-22.66	-96.74	-15.15	-277.37	-43.05	-48.88	-29.38
FVSBN	**-13.17**	-12.39	-83.64	-10.27	-276.88	-39.30	-49.84	-29.35
Chow–Liu	-18.51	-20.57	-87.72	-20.99	-281.01	-48.87	-55.60	-33.92
MADE	**-13.12**	**-11.90**	-83.63	-9.68	-280.25	-28.34	-47.10	-28.53
MADE-E	**-13.13**	**-11.90**	**-79.66**	-9.69	-277.28	-30.04	-46.74	**-28.25**
NADE	**-13.19**	-11.99	-84.81	-9.81	**-273.08**	**-27.22**	-46.66	-28.39
NADE-E	**-13.19**	-12.58	-82.31	-9.69	**-272.39**	**-27.32**	**-46.12**	**-27.87**

Table 2: Average log-likelihood performance of tractable distribution baselines and NADE models, on binary vector data sets. The best result is shown in bold, along with any other result with an overlapping confidence interval.

Table 2 presents the results. We observe that NADE restricted to a fixed ordering of the inputs achieves very competitive performance compared to the baselines. However, the order-agnostic version of NADE is overall the best method, being among the top performing model for 5 data sets out of 8.

The performance of fixed-order NADE is surprisingly robust to variations of the chosen input ordering. The standard deviation on the average log-likelihood when varying the ordering was small: on Mushrooms, DNA and NIPS-0-12, we observed standard deviations of 0.045, 0.05 and 0.15, respectively. However, models with different orders can do well on different test examples, which explains why ensembling can still help.

7.2 Binary Image Data Set

We now consider the case of an image data set, constructed by binarizing the MNIST digit data set. Each image has been stochastically binarized according to their pixel intensity as generated by Salakhutdinov and Murray (2008). This benchmark has been a popular choice for the evaluation of generative neural network models. Here, we investigate two questions:

1. How does NADE compare to intractable generative models?

2. Does the use of a convolutional architecture improve the performance of NADE?

For these experiments, in addition to the baselines already described in Section 7.1, we consider the following:

- **DARN** (Gregor et al., 2014): This deep generative autoencoder has two hidden layers, one deterministic and one with binary stochastic units. Both layers have 500 units (denoted as $n_h = 500$). Adaptive weight noise (adaNoise) was either used or not to avoid the need for early stopping (Graves, 2011). Evaluation of exact test probabilities is intractable for large latent representations. Hence, Monte Carlo was used to approximate the expected description length, which corresponds to an upper bound on the negative log-likelihood.

Model	$-\log p$	$\approx$
MoBernoullis K=10	168.95	
MoBernoullis K=500	137.64	
Chow–Liu tree	134.99	
MADE 2hl (32 masks)	86.64	
RBM (500 h, 25 CD steps)		86.34
DBN 2hl		84.55
DARN $n_h = 500$		84.71
DARN $n_h = 500$ (adaNoise)		**84.13**
NADE (fixed order)	88.33	
DeepNADE 1hl (no input masks)	99.37	
DeepNADE 2hl (no input masks)	95.33	
DeepNADE 1hl	92.17	
DeepNADE 2hl	89.17	
DeepNADE 3hl	89.38	
DeepNADE 4hl	89.60	
EoNADE 1hl (2 orderings)	90.69	
EoNADE 1hl (128 orderings)	87.71	
EoNADE 2hl (2 orderings)	87.96	
EoNADE 2hl (128 orderings)	**85.10**	

Table 3: Negative log-likelihood test results of models ignorant of the 2D topology on the binarized MNIST data set.

- **DRAW** (Gregor et al., 2015): Similar to a variational autoencoder where both the encoder and the decoder are LSTMs, guided (or not) by an attention mechanism. In this model, both LSTMs (encoder and decoder) are composed of 256 recurrent hidden units and always perform 64 timesteps. When the attention mechanism is enabled, patches (2×2 pixels) are provided as inputs to the encoder instead of the whole image and the decoder also produces patches (5×5 pixels) instead of a whole image.

- **Pixel RNN** (Oord et al., 2016): NADE-like model for natural images that is based on convolutional and LSTM hidden units. This model has 7 hidden layers, each composed of 16 units. Oord et al. (2016) proposed a novel two-dimensional LSTM, named Diagonal BiLSTM, which is used in this model. Unlike our ConvNADE, the ordering is fixed before training and at test time, and corresponds to a scan of the image in a diagonal fashion starting from a corner at the top and reaching the opposite corner at the bottom.

We compare these baselines with some NADE variants. The performance of a basic (fixed-order, single hidden layer) NADE model is provided in Table 3 and samples are illustrated in Figure 4. More importantly, we will focus on whether the following variants achieve better test set performance:

- **DeepNADE**: Multiple layers (1hl, 2hl, 3hl or 4hl) trained according to the order-agnostic procedure described in Section 4. Information about which inputs are masked

was either provided or not (no input masks) to the model. The rectified linear activation function was used for all hidden layers. Minibatch gradient descent was used for training, with minibatches of size 1000. Training consisted of 200 iterations of 1000 parameter updates. Each hidden layer was pre-trained according to Algorithm 2. We report an average of the average test log-likelihoods over ten different random orderings.

- **EoNADE**: This variant is similar to DeepNADE except for the log-likelihood on the test set, which is instead computed from an ensemble that averages predictive probabilities over 2 or 128 orderings. To clarify, the DeepNADE results report the typical performance of one ordering, by averaging results after taking the log, and so do not combine the predictions of the models like EoNADE does.

- **ConvNADE**: Multiple convolutional layers trained according to the order-agnostic procedure described in Section 4. The exact architecture is shown in Figure 5(a). Information about which inputs are masked was either provided or not (no input masks). The rectified linear activation function was used for all hidden layers. The Adam optimizer (Kingma and Ba, 2015) was used with a learning rate of 10^{-4}. Early stopping was used with a look ahead of 10 epochs, using Equation 34 to get a stochastic estimate of the validation set average log-likelihood. An ensemble using 128 orderings was used to compute the log-likelihood on the test set.

- **ConvNADE + DeepNADE**: This variant is similar to ConvNADE except for the aggregation of a separate DeepNADE model at the end of the network. The exact architecture is shown in Figure 5(b). The training procedure is the same as with ConvNADE.

Algorithm 2 Pre-training of a NADE with n hidden layers on data set X.

```
procedure PRETRAIN(n, X)
    if n = 1 then
        return RANDOM-ONE-HIDDEN-LAYER-NADE
    else
        nade ← PRETRAIN(n − 1, X)
        nade ← REMOVE-OUTPUT-LAYER(nade)
        nade ← ADD-A-NEW-HIDDEN-LAYER(nade)
        nade ← ADD-A-NEW-OUTPUT-LAYER(nade)
        nade ← TRAIN-ALL(nade,X,iters=20)        ▷ Train for 20 iterations.
        return nade
    end if
end procedure
```

Table 3 presents the results obtained by models ignorant of the 2D topology, such as the basic NADE model. Addressing the first question, we observe that the order-agnostic version of NADE with two hidden layers is competitive with intractable generative models. Moreover, examples of the ability of DeepNADE to solve inference tasks by marginalization and conditional sampling are shown in Figure 6.

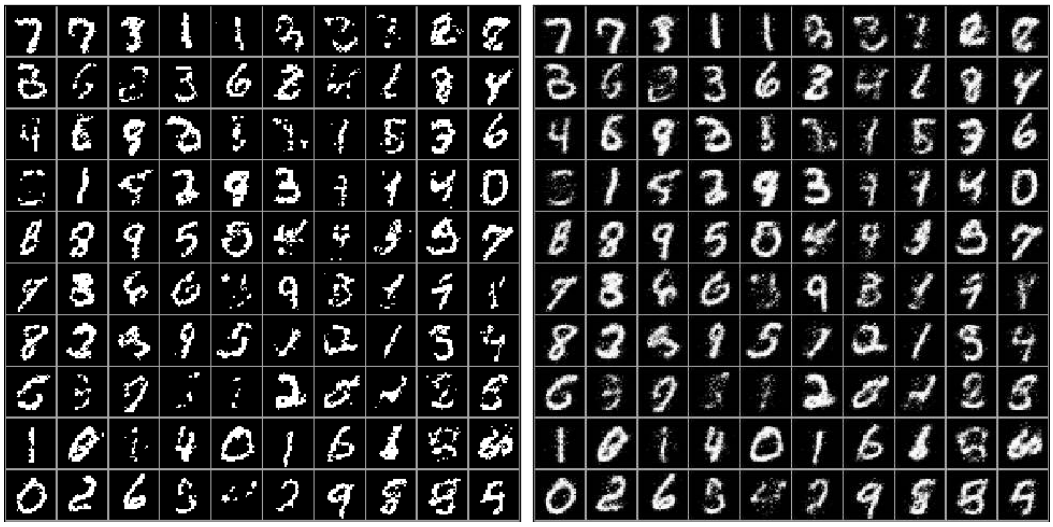


Figure 4: **Left**: samples from NADE trained on binarized MNIST. **Right**: probabilities from which each pixel was sampled. Ancestral sampling was used with the same fixed ordering used during training.

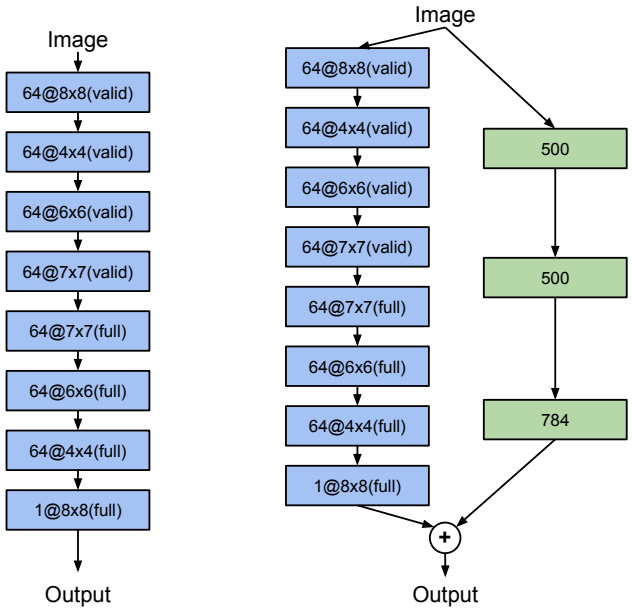
Model	$-\log p$	$\leq$
DRAW (without attention)		87.40
DRAW		80.97
Pixel RNN	**79.20**	
ConvNADE+DeepNADE (no input masks)	85.25	
ConvNADE	81.30	
ConvNADE+DeepNADE	80.82	

Table 4: Negative log-likelihood test results of models exploiting 2D topology on the binarized MNIST data set.

Now, addressing the second question, we can see from Table 4 that convolutions do improve the performance of NADE. Moreover, we observe that providing information about which inputs are masked is essential to obtaining good results. We can also see that combining convolutional and fully-connected layers helps. Even though ConvNADE+DeepNADE performs slightly worse than Pixel RNN, we note that our proposed approach is order-agnostic, whereas Pixel RNN requires a fixed ordering. Figure 7 shows samples obtained from the ConvNADE+DeepNADE model using ancestral sampling on a random ordering.

7.3 Real-Valued Observations Data Sets

In this section, we compare the statistical performance of RNADE to mixtures of Gaussians (MoG) and factor analyzers (MFA), which are surprisingly strong baselines in some tasks (Tang et al., 2012; Zoran and Weiss, 2012).



(a) ConvNADE (b) ConvNADE + DeepNADE

Figure 5: Network architectures for binarized MNIST. (a) ConvNADE with 8 convolutional layers (depicted in blue). The number of feature maps for a given layer is given by the number before the "@" symbol followed by the filter size and the type of convolution is specified in parentheses. (b) The same ConvNADE combined with a DeepNADE consisting of three fully-connected layers of respectively 500, 500 and 784 units.

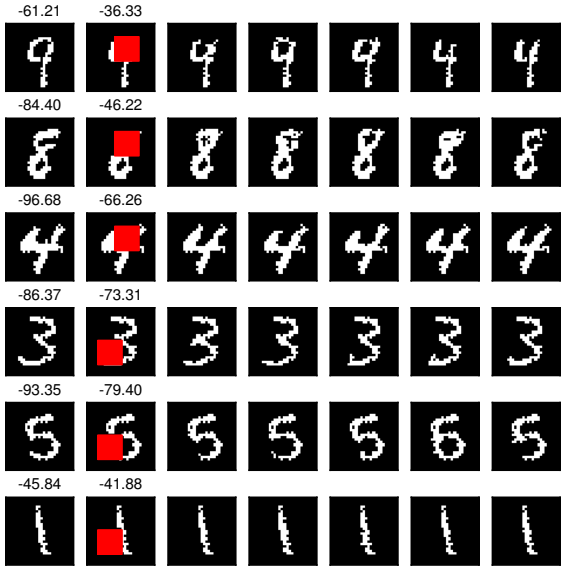


Figure 6: Example of marginalization and sampling. The first column shows five examples from the test set of the MNIST data set. The second column shows the density of these examples when a random 10×10 pixel region is marginalized. The right-most five columns show samples for the hollowed region. Both tasks can be done easily with a NADE where the pixels to marginalize are at the end of the ordering.

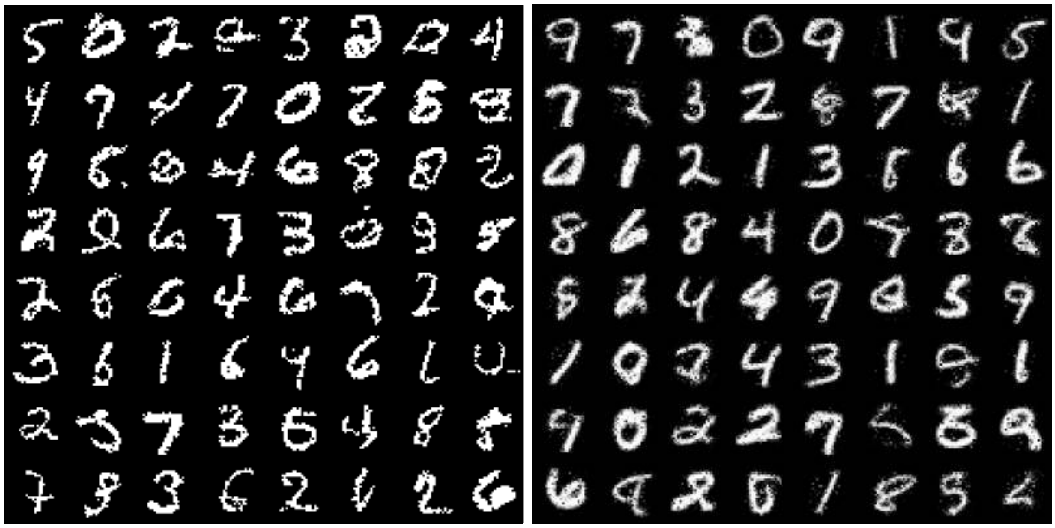


Figure 7: **Left**: samples from ConvNADE+DeepNADE trained on binarized MNIST. **Right**: probabilities from which each pixel was sampled. Ancestral sampling was used with a different random ordering for each sample.

7.3.1 Low-dimensional data

We start by considering three UCI data sets (Bache and Lichman, 2013), previously used to study the performance of other density estimators (Silva et al., 2011; Tang et al., 2012), namely: *red wine*, *white wine* and *parkinsons*. These are low dimensional data sets (see Table 5) with hard thresholds and non-linear dependencies that make it difficult to fit mixtures of Gaussians or factor analyzers.

Following Tang et al. (2012), we eliminated discrete-valued attributes and an attribute from every pair with a Pearson correlation coefficient greater than 0.98. We normalized each dimension of the data by subtracting its training-subset sample mean and dividing by its standard deviation. All results are reported on the normalized data.

We use full-covariance Gaussians and mixtures of factor analysers as baselines. Models were compared on their log-likelihood on held-out test data. Due to the small size of the data sets (see Table 5), we used 10-folds, using 90% of the data for training, and 10% for testing.

We chose the hyperparameter values for each model by doing per-fold cross-validation, using a ninth of the training data as validation data. Once the hyperparameter values have been chosen, we train each model using all the training data (including the validation data) and measure its performance on the 10% of held-out testing data. In order to avoid overfitting, we stopped the training after reaching a training likelihood higher than the one obtained on the best validation-wise iteration of the best validation run. Early stopping was important to avoid overfitting the RNADE models. It also improved the results of the MFAs, but to a lesser degree.

The MFA models were trained using the EM algorithm (Ghahramani and Hinton, 1996; Verbeek, 2005). We cross-validated the number of components and factors. We also selected the number of factors from $2, 4, \dots D$, where choosing D results in a mixture of Gaussians, and the number of components was chosen among $2, 4, \dots 50$. Cross-validation selected fewer than 50 components in every case.

We report the performance of several RNADE models using different parametric forms for the one-dimensional conditionals: Gaussian with fixed variance (RNADE-FV), Gaussian with variable variance (RNADE-Gaussian), *sinh-arcsinh* distribution (RNADE-SAS), mixture of Gaussians (RNADE-MoG), and mixture of Laplace distributions (RNADE-MoL). All RNADE models were trained by stochastic gradient descent, using minibatches of size 100, for 500 epochs, each epoch comprising 10 minibatches. We fixed the number of hidden units to 50, and the non-linear activation function of the hidden units to ReLU. Three hyperparameters were cross-validated using grid-search: the number of components on each one-dimensional conditional (only applicable to the RNADE-MoG and RNADE-MoL models) was chosen from $\{2, 5, 10, 20\}$, the weight-decay (used only to regularize the input to hidden weights) from $\{2.0, 1.0, 0.1, 0.01, 0.001, 0\}$, and the learning rate from $\{0.1, 0.05, 0.025, 0.0125\}$. Learning rates were decreased linearly to reach 0 after the last epoch.

The results are shown in Table 6. RNADE with mixture of Gaussian conditionals was among the statistically significant group of best models on all data sets. As shown in Figure 8, RNADE-SAS and RNADE-MoG models are able to capture hard thresholds and heteroscedasticity.

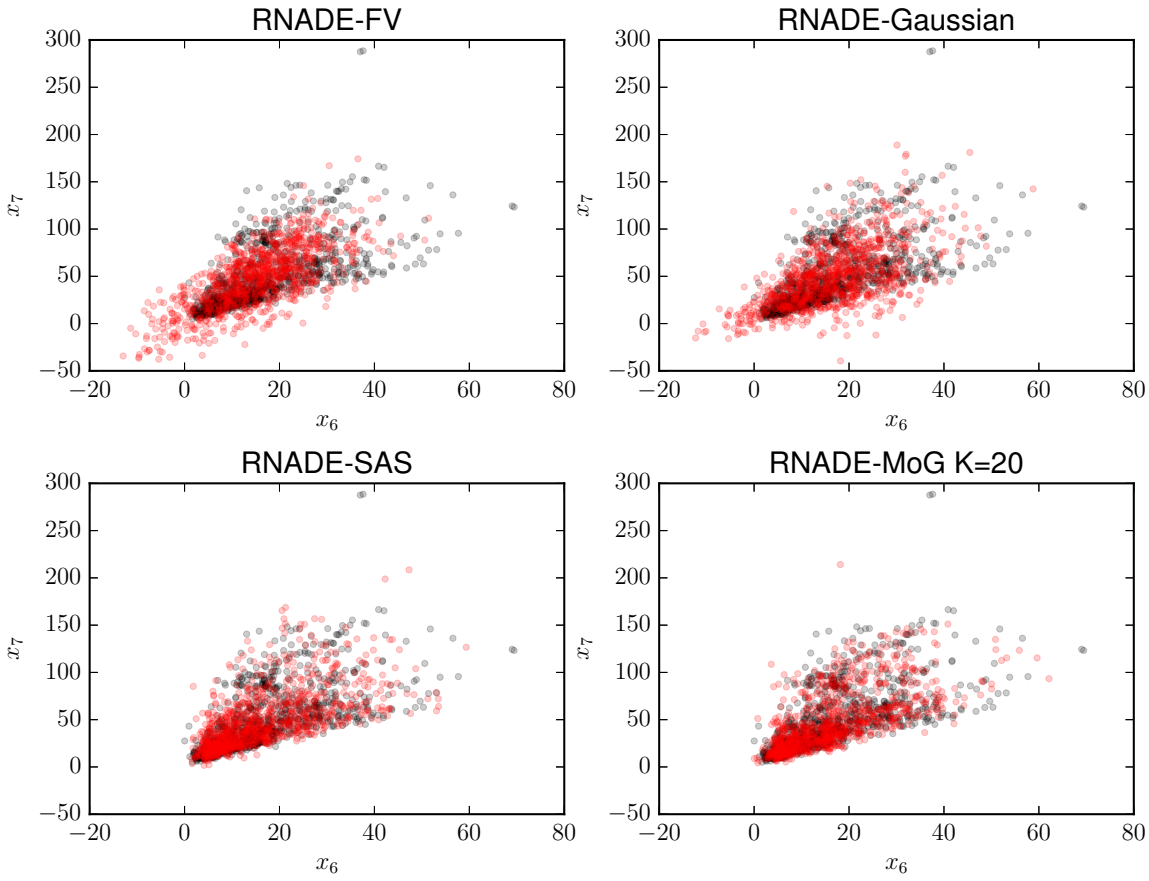


Figure 8: Scatter plot of dimensions x_7 vs x_6 of the *red wine* data set. A thousand data points from the data set are shown in black in all subfigures. As can be observed, this conditional distribution $p(x_7 \mid x_6)$ is heteroscedastic, skewed and has hard thresholds. In red, a thousand samples from four RNADE models with different one-dimensional conditional forms are shown. **Top-left**: In red, one thousand samples from a RNADE-FV model. **Top-right**: In red, one thousand samples from a RNADE-Gaussian model. **Bottom-left**: In red, one thousand samples from a RNADE-SAS (sinh-arcsinh distribution) model. **Bottom-right**: In red, one thousand samples from a RNADE-MoG model with 20 components per one-dimensional conditional. The RNADE-SAS and RNADE-MoG models successfully capture all the characteristics of the data.

	Red wine	White wine	Parkinsons
Dimensionality	11	11	15
Total number of data points	1599	4898	5875

Table 5: Dimensionality and size of the UCI data sets used in Section 7.3.1

Model	Red wine	White wine	Parkinsons
Gaussian	−13.18	−13.20	−10.85
MFA	−10.19	−10.73	−1.99
RNADE-FV	−12.29	−12.50	−8.87
RNADE-Gaussian	−11.99	−12.20	−3.47
RNADE-SAS	−9.86	−11.22	−3.07
RNADE-MoG	**−9.36**	**−10.23**	**−0.90**
RNADE-MoL	**−9.46**	−10.38	−2.63

Table 6: Average test set log-likelihoods per data point for seven models on three UCI data sets. Performances not in bold can be shown to be significantly worse than at least one of the results in bold as per a paired t-test on the ten mean-likelihoods (obtained from each data fold), with significance level 0.05.

7.3.2 Natural image patches

We also measured the ability of RNADE to model small patches of natural images. Following the work of Zoran and Weiss (2011), we use 8-by-8-pixel patches of monochrome natural images, obtained from the BSDS300 data set (Martin et al., 2001; Figure 9 gives examples).

Pixels in this data set can take a finite number of brightness values ranging from 0 to 255. We added uniformly distributed noise between 0 and 1 to the brightness of each pixel. We then divided by 256, making the pixels take continuous values in the range $[0, 1]$. Adding noise prevents deceivingly high-likelihood solutions that assign narrow high-density spikes around some of the possible discrete values.

We subtracted the mean pixel value from each patch. Effectively reducing the dimensionality of the data. Therefore we discarded the 64th (bottom-right) pixel, which would be perfectly predictable and models could fit arbitrarily high densities to it. All of the results in this section were obtained by fitting the pixels in a raster-scan order.

Experimental details follow. We trained our models by using patches randomly drawn from 180 images in the training subset of BSDS300. We used the remaining 20 images in the training subset as validation data. We used 1000 random patches from the validation subset to early-stop training of RNADE. We measured the performance of each model by their log-likelihood on one million patches drawn randomly from the test subset of 100 images not present in the training data. Given the larger scale of this data set, hyperparameters of the RNADE and MoG models were chosen manually using the performance of preliminary runs on the validation data, rather than by grid search.

All RNADE models reported use ReLU activations for the hidden units. The RNADE models were trained by stochastic gradient descent, using 25 data points per minibatch, for a total of 1,000 epochs, each comprising 1,000 minibatches. The learning rate was initialized to 0.001, and linearly decreased to reach 0 after the last epoch. Gradient momentum with factor 0.9 was used, but initiated after the first epoch. A weight decay rate of 0.001 was applied to the input-to-hidden weight matrix only. We found that multiplying the gradient of the mean output parameters by the standard deviation improves results of the models with mixture outputs[5]. RNADE training was early stopped but didn't show signs of overfitting. Even larger models might perform better.

The MoG models were trained using 1,000 iterations of minibatch EM. At each iteration 20,000 randomly sampled data points were used in an EM update. A step was taken from the previous parameters' value towards the parameters resulting from the M-step: $\boldsymbol{\theta_t} = (1 - \eta)\boldsymbol{\theta_{t-1}} + \eta\boldsymbol{\theta_{EM}}$. The step size, η, was scheduled to start at 0.1 and linearly decreased to reach 0 after the last update. The training of the MoG was early-stopped and also showed no signs of overfitting.

The results are shown in Table 7. We report the average log-likelihood of each model for a million image patches from the test set. The ranking of RNADE models is maintained when ordered by validation likelihood: the model with best test-likelihood would have been chosen using crossvalidation across all the RNADE models shown in the table. We also compared RNADE with a MoG trained by Zoran and Weiss (downloaded from Daniel Zoran's website) from which we removed the 64th row and column of each covariance matrix. There are two differences in the set-up of our experiments and those of Zoran and Weiss. First, we learned the means of the MoG components, while Zoran and Weiss (2011) fixed them to zero. Second, we held-out 20 images from the training set to do early-stopping and hyperparameter optimisation, while they used the 200 images for training.

The RNADE-FV model with fixed conditional variances obtained very low statistical performance. Adding an output parameter per dimension to have variable standard deviations made our models competitive with MoG with 100 full-covariance components. However, in order to obtain results superior to the mixture of Gaussians model trained by Zoran and Weiss, we had to use richer conditional distributions: one-dimensional mixtures of Gaussians (RNADE-MoG). On average, the best RNADE model obtained 3.3 nats per patch higher log-density than a MoG fitted with the same training data.

In Figure 9, we show one hundred examples from the test set, one hundred examples from Zoran and Weiss' mixture of Gaussians, and a hundred samples from our best RNADE-MoG model. Similar patterns can be observed in the three cases: uniform patches, edges, and locally smooth noisy patches.

7.3.3 Speech acoustics

We also measured the ability of RNADE to model small patches of speech spectrograms, extracted from the TIMIT data set (Garofolo et al., 1993). The patches contained 11 frames of 20 filter-banks plus energy; totalling 231 dimensions per data point. A good generative model of speech acoustics could be used, for example, in denoising, or speech detection tasks.

5. Empirically, we found this to work better than regular gradients and also better than multiplying by the variances, which would provide a step with the right units.

Model	Test log-likelihood
MoG $K=200$ (Zoran and Weiss, 2012)[a]	152.8
MoG $K=100$	144.7
MoG $K=200$	150.4
MoG $K=300$	150.4
RNADE-FV $h=512$	100.3
RNADE-Gaussian $h=512$	143.9
RNADE-Laplace $h=512$	145.9
RNADE-SAS[b] $h=512$	148.5
RNADE-MoG $K=2$ $h=512$	149.5
RNADE-MoG $K=2$ $h=1024$	150.3
RNADE-MoG $K=5$ $h=512$	152.4
RNADE-MoG $K=5$ $h=1024$	152.7
RNADE-MoG $K=10$ $h=512$	153.5
RNADE-MoG $K=10$ $h=1024$	**153.7**
RNADE-MoL $K=2$ $h=512$	149.3
RNADE-MoL $K=2$ $h=1024$	150.1
RNADE-MoL $K=5$ $h=512$	151.5
RNADE-MoL $K=5$ $h=1024$	151.4
RNADE-MoL $K=10$ $h=512$	152.3
RNADE-MoL $K=10$ $h=1024$	152.5

Table 7: Average per-example log-likelihood of several mixture of Gaussian and RNADE models on 8×8 pixel patches of natural images. These results are reported in nats and were calculated using one million patches. Standard errors due to the finite test sample size are lower than 0.1 nats in every case. h indicates the number of hidden units in the RNADE models, and K the number of one-dimensional components for each conditional in RNADE or the number of full-covariance components for MoG.

a. This model was trained using the full 200 images in the BSDS training data set, the rest of the models were trained using 180, reserving 20 for hyperparameter crossvalidation and early-stopping.

b. Training an RNADE with sinh-arcsinh conditionals required the use of a starting learning rate 20 times smaller to avoid divergence during training. For this reason, this model was trained for 2000 epochs.

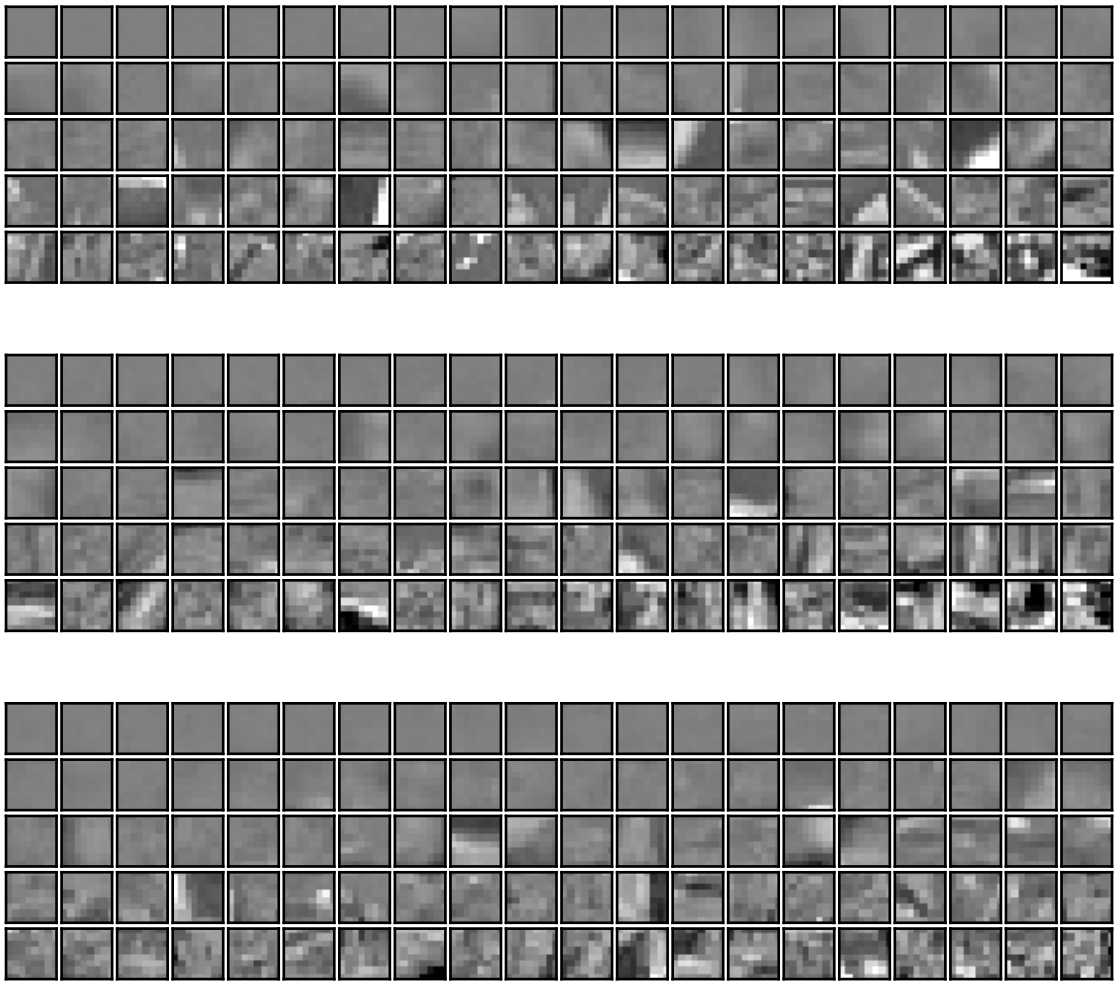


Figure 9: **Top:** 100 8×8 patches from the BSDS test set. **Center:** 100 samples from a mixture of Gaussians with 200 full-covariance components. **Bottom:** 100 samples from an RNADE with 1024 hidden units and 10 Gaussian components per conditional. All data and samples were drawn randomly and sorted by their density under the RNADE.

Model	Test LogL
MoG $N=50$	110.4
MoG $N=100$	112.0
MoG $N=200$	112.5
MoG $N=300$	112.5
RNADE-Gaussian	110.6
RNADE-Laplace	108.6
RNADE-SAS	119.2
RNADE-MoG $K=2$	121.1
RNADE-MoG $K=5$	124.3
RNADE-MoG $K=10$	**127.8**
RNADE-MoL $K=2$	116.3
RNADE-MoL $K=5$	120.5
RNADE-MoL $K=10$	123.3

Table 8: Log-likelihood of several MoG and RNADE models on the core-test set of TIMIT measured in nats. Standard errors due to the finite test sample size are lower than 0.4 nats in every case. RNADE obtained a higher (better) log-likelihood.

We fitted the models using the standard TIMIT training subset, which includes recordings from 605 speakers of American English. We compare RNADE with a mixture of Gaussians by measuring their log-likelihood on the complete TIMIT core-test data set: a held-out set of 25 speakers.

The RNADE models have 512 hidden units, ReLU activations, and a mixture of 20 one-dimensional Gaussian components per output. Given the large scale of this data set, hyperparameter choices were again made manually using validation data. The same training procedures for RNADE and mixture of Gaussians were used as for natural image patches.

The RNADE models were trained by stochastic gradient descent, with 25 data points per minibatch, for a total of 200 epochs, each comprising 1,000 minibatches. The learning rate was initialized to 0.001 and linearly decreased to reach 0 after the last epoch. Gradient momentum with momentum factor 0.9 was used, but initiated after the first epoch. A weight decay rate of 0.001 was applied to the input-to-hidden weight matrix only. Again, we found that multiplying the gradient of the mean output parameters by the standard deviation improved results. RNADE training was early stopped but didn't show signs of overfitting.

As for the MoG model, it was trained exactly as in Section 7.3.2.

The results are shown in Table 8. The best RNADE (which would have been selected based on validation results) has 15 nats higher likelihood per test example than the best mixture of Gaussians. Examples from the test set, and samples from the MoG and RNADE-MoG models are shown in Figure 10. In contrast with the log-likelihood measure, there are no marked differences between the samples from each model. Both sets of samples look like blurred spectrograms, but RNADE seems to capture sharper formant structures (peaks of energy at the lower frequency bands characteristic of vowel sounds).

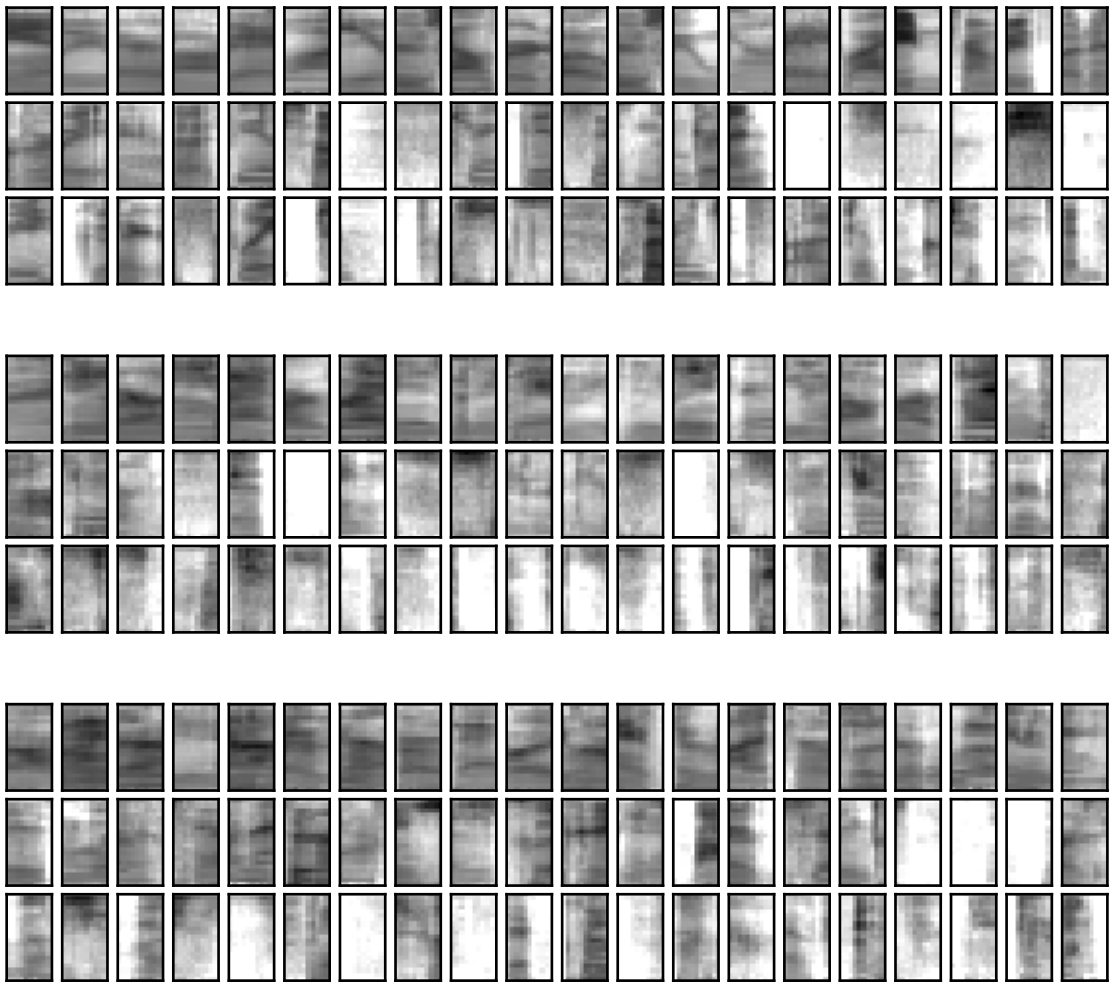


Figure 10: **Top:** 60 data points from the TIMIT core-test set. **Center:** 60 samples from a MoG model with 200 components. **Bottom:** 60 samples from an RNADE with 10 Gaussian output components per dimension. For each data point displayed, time is shown on the horizontal axis, the bottom row displays the energy feature, while the others display the Mel filter bank features (in ascending frequency order from the bottom). All data and samples were drawn randomly and sorted by density under the RNADE model.

8. Conclusion

We've described the Neural Autoregressive Distribution Estimator, a tractable, flexible and competitive alternative to directed and undirected graphical models for unsupervised distribution estimation.

Since the publication of the first formulation of NADE (Larochelle and Murray, 2011), it has been extended to many more settings, other than those described in this paper. Larochelle and Lauly (2012); Zheng et al. (2015b) adapted NADE for topic modeling of documents and images, while Boulanger-Lewandowski et al. (2012) used NADE for modeling music sequential data. Theis and Bethge (2015) and Oord et al. (2016) proposed different NADE models for images than the one we presented, applied to natural images and based on convolutional and LSTM hidden units. Zheng et al. (2015a) used a NADE model to integrate an attention mechanism into an image classifier. Bornschein and Bengio (2015) showed that NADE could serve as a powerful prior over the latent state of directed graphical model. These are just a few examples of many possible ways one can leverage the flexibility and effectiveness of NADE models.

References

Kevin Bache and Moshe Lichman. UCI machine learning repository, 2013. `http://archive.ics.uci.edu/ml`.

Yoshua Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

Yoshua Bengio and Samy Bengio. Modeling high-dimensional discrete data with multi-layer neural networks. In *Advances in Neural Information Processing Systems 12*, pages 400–406. MIT Press, 2000.

Julian Besag. Statistical analysis of non-lattice data. *The Statistician*, 24(3):179–195, 1975.

Christopher M. Bishop. Mixture density networks. Technical Report NCRG 4288, Neural Computing Research Group, Aston University, Birmingham, 1994.

Jörg Bornschein and Yoshua Bengio. Reweighted wake-sleep. In *Proceedings of the 3rd International Conference on Learning Representations*. arXiv:1406.2751, 2015.

Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In *Proceedings of the 29th International Conference on Machine Learning*, pages 1159–1166. Omnipress, 2012.

Yuri Burda, Ruslan Salakhutdinov, and Roger Grosse. Importance weighted autoencoders. In *Proceedings of the 4th International Conference on Learning Representations*. arXiv:1509.00519v3, 2016.

KyungHyun Cho, Tapani Raiko, and Alexander Ilin. Parallel tempering is efficient for learning restricted Boltzmann machines. In *Proceedings of the International Joint Conference on Neural Networks*. IEEE, 2010.

KyungHyun Cho, Tapani Raiko, and Alexander Ilin. Enhanced gradient for training restricted Boltzmann machines. *Neural Computation*, 25:805–31, 2013.

C.K. Chow and C.N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.

George E. Dahl, Tara N. Sainath, and Geoffrey E. Hinton. Improving deep neural networks for LVCSR using rectified linear units and dropout. *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8609–8613, 2013.

Peter Dayan, Geoffrey E. Hinton, Radford M. Neal, and Richard S. Zemel. The Helmholtz machine. *Neural Computation*, 7:889–904, 1995.

Emily L. Denton, Soumith Chintala, Arthur Szlam, and Rob Fergus. Deep generative image models using a Laplacian pyramid of adversarial networks. In *Advances in Neural Information Processing Systems 28*, pages 1486–1494. Curran Associates, Inc., 2015.

Guillaume Desjardins, Aaron Courville, Yoshua Bengio, Pascal Vincent, and Olivier Delalleau. Tempered Markov chain Monte Carlo for training of restricted Boltzmann machine. *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics, JMLR W&CP*, 9:145–152, 2010.

Yoav Freund and David Haussler. Unsupervised learning of distributions on binary vectors using two layer networks. In *Advances in Neural Information Processing Systems 4*, pages 912–919. Morgan-Kaufmann, 1992.

Brendan J. Frey, Geoffrey E. Hinton, and Peter Dayan. Does the wake-sleep algorithm learn good density estimators? In *Advances in Neural Information Processing Systems 8*, pages 661–670. MIT Press, 1996.

J. Garofolo, L. Lamel, W. Fisher, J. Fiscus, D. Pallett, N. Dahlgren, and V. Zue. DARPA TIMIT acoustic-phonetic continuous speech corpus CD-ROM. NIST, 1993.

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. MADE: Masked autoencoder for distribution estimation. *Proceedings of the 32nd International Conference on Machine Learning, JMLR W&CP*, 37:881–889, 2015.

Zoubin Ghahramani and Geoffrey E. Hinton. The EM algorithm for mixtures of factor analyzers. Technical Report CRG-TR-96-1, University of Toronto, 1996.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*, pages 2672–2680, 2014.

Alex Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems 24*, pages 2348–2356. Curran Associates, Inc., 2011.

Karol Gregor and Yann LeCun. Learning representations by maximizing compression. Technical report, arXiv:1108.1169, 2011.

Karol Gregor, Andriy Mnih, and Daan Wierstra. Deep autoregressive networks. *Proceedings of the 31st International Conference on Machine Learning, JMLR W&CP*, 32:1242–1250, 2014.

Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. DRAW: a recurrent neural network for image generation. *Proceedings of the 32nd International Conference on Machine Learning, JMLR W&CP*, 37:1462–1471, 2015.

Arthur Gretton, Karsten M. Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alex J. Smola. A kernel method for the two-sample-problem. In *Advances in Neural Information Processing Systems 19*, pages 513–520. MIT Press, 2007.

Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 297–304, 2010.

Stefan Harmeling and Christopher K.I. Williams. Greedy learning of binary latent trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(6):1087–1097, 2011.

Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:1771–1800, 2002.

Geoffrey E. Hinton, Peter Dayan, Brendan J. Frey, and Radford M. Neal. The wake-sleep algorithm for unsupervised neural networks. *Science*, 268:1161–1558, 1995.

Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

Aapo Hyvärinen. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6:695–709, 2005.

Aapo Hyvärinen. Some extensions of score matching. *Computational Statistics and Data Analysis*, 51:2499–2512, 2007a.

Aapo Hyvärinen. Connections between score matching, contrastive divergence, and pseudo-likelihood for continuous-valued variables. *IEEE Transactions on Neural Networks*, 18: 1529–1531, 2007b.

Diederik P. Kingma and Jimmy Lei Ba. Adam: a method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations*. arXiv:1412.6980v5, 2015.

Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. In *Proceedings of the 2nd International Conference on Learning Representations*. arXiv:1312.6114v10, 2014.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

Hugo Larochelle and Stanislas Lauly. A neural autoregressive topic model. In *Advances in Neural Information Processing Systems 25*, pages 2708–2716. Curran Associates, Inc., 2012.

Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics, JMLR W&CP*, 15:29–37, 2011.

Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. Deep learning. *Nature*, 521(7553): 436–444, 2015.

Yujia Li, Kevin Swersky, and Richard S. Zemel. Generative moment matching networks. *Proceedings of the 32nd International Conference on Machine Learning, JMLR W&CP*, 37:1718–1727, 2015.

Benjamin Marlin, Kevin Swersky, Bo Chen, and Nando de Freitas. Inductive principles for restricted Boltzmann machine learning. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010.

D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *International Conference on Computer Vision*, volume 2, pages 416–423. IEEE, July 2001.

Grégoire Montavon and Klaus-Robert Müller. Deep Boltzmann machines and the centering trick. In *Neural Networks: Tricks of the Trade, Second Edition*, pages 621–637. Springer, 2012.

Radford M. Neal. Connectionist learning of belief networks. *Artificial Intelligence*, 56:71–113, 1992.

Jiquan Ngiam, Zhenghao Chen, Pang Wei Koh, and Andrew Y. Ng. Learning deep energy models. In *Proceedings of the 28th International Conference on Machine Learning*, pages 1105–1112. Omnipress, 2011.

Aäron Van Den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *Proceedings of the 33rd International Conference on Machine Learning, JMLR W&CP*, 2016. To appear. arXiv:1601.06759v2.

Dirk Ormoneit and Volker Tresp. Improved Gaussian mixture density estimates using Bayesian penalty terms and network averaging. In *Advances in Neural Information Processing Systems 8*, pages 542–548. MIT Press, 1995.

Tapani Raiko, Li Yao, Kyunghyun Cho, and Yoshua Bengio. Iterative neural autoregressive distribution estimator (NADE-k). In *Advances in Neural Information Processing Systems 27*, pages 325–333. Curran Associates, Inc., 2014.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *Proceedings of the 31st International Conference on Machine Learning, JMLR W&CP*, 32:1278–1286, 2014.

Ruslan Salakhutdinov. Learning in Markov random fields using tempered transitions. In *Advances in Neural Information Processing Systems 22*, pages 1598–1606. Curran Associates, Inc., 2009.

Ruslan Salakhutdinov. Learning deep Boltzmann machines using adaptive MCMC. In *Proceedings of the 27th International Conference on Machine Learning*, pages 943–950. Omnipress, 2010.

Ruslan Salakhutdinov and Geoffrey E. Hinton. Deep Boltzmann machines. *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, JMLR W&CP*, 5:448–455, 2009.

Ruslan Salakhutdinov and Hugo Larochelle. Efficient learning of deep Boltzmann machines. *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics, JMLR W&CP*, 9:693–700, 2010.

Ruslan Salakhutdinov and Iain Murray. On the quantitative analysis of deep belief networks. In *Proceedings of the 25th International Conference on Machine Learning*, pages 872–879. Omnipress, 2008.

Ricardo Silva, Charles Blundell, and Yee Whye Teh. Mixed cumulative distribution networks. *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics, JMLR W&CP*, 15:670–678, 2011.

Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. In D.E. Rumelhart and J.L. McClelland, editors, *Parallel Distributed Processing: Volume 1: Foundations*, volume 1, chapter 6, pages 194–281. MIT Press, Cambridge, 1986.

Padhraic Smyth and David Wolpert. Linearly combining density estimators via stacking. *Machine Learning*, 36(1-2):59–83, 1999.

Jascha Sohl-Dickstein, Peter Battaglino, and Michael R. DeWeese. Minimum probability flow learning. In *Proceedings of the 28th International Conference on Machine Learning*, pages 905–912. Omnipress, 2011.

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: the all convolutional net. In *Proceedings of the 3rd International Conference on Learning Representations*. arXiv:1412.6806v3, 2015.

Yichuan Tang, Ruslan Salakhutdinov, and Geoffrey E. Hinton. Deep mixtures of factor analysers. In *Proceedings of the 29th International Conference on Machine Learning*, pages 505–512. Omnipress, 2012.

The Theano Development Team, Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, et al. Theano: A python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*, 2016.

Lucas Theis and Matthias Bethge. Generative image modeling using spatial lstms. In *Advances in Neural Information Processing Systems 28*, pages 1927–1935. Curran Associates, Inc., 2015.

Tijmen Tieleman. Training restricted Boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1064–1071. Omnipress, 2008.

Tijmen Tieleman and Geoffrey E. Hinton. Using fast weights to improve persistent contrastive divergence. In *Proceedings of the 26th International Conference on Machine Learning*, pages 1033–1040. Omnipress, 2009.

Benigno Uria. *Connectionist multivariate density-estimation and its application to speech synthesis*. PhD thesis, The University of Edinburgh, 2015.

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. Curran Associates, Inc., 2013.

Benigno Uria, Iain Murray, and Hugo Larochelle. A deep and tractable density estimator. *Proceedings of the 31st International Conference on Machine Learning, JMLR W&CP*, 32: 467–475, 2014.

Jakob Verbeek. Mixture of factor analyzers Matlab implementation, 2005. `http://lear.inrialpes.fr/~verbeek/software.php`.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1096–1103. Omnipress, 2008.

Max Welling, Michal Rosen-Zvi, and Geoffrey E. Hinton. Exponential family harmoniums with an application to information retrieval. In *Advances in Neural Information Processing Systems 17*, pages 1481–1488. MIT Press, 2005.

Laurent Younes. Parameter inference for imperfectly observed Gibbsian fields. *Probability Theory Related Fields*, 82:625–645, 1989.

Yin Zheng, Richard S. Zemel, Yu-Jin Zhang, and Hugo Larochelle. A neural autoregressive approach to attention-based recognition. *International Journal of Computer Vision*, 113 (1):67–79, 2015a.

Yin Zheng, Yu-Jin Zhang, and Hugo Larochelle. A deep and autoregressive approach for topic modeling of multimodal data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(6):1056–1069, 2015b.

Daniel Zoran and Yair Weiss. From learning models of natural image patches to whole image restoration. In *International Conference on Computer Vision*, pages 479–486. IEEE, 2011.

Daniel Zoran and Yair Weiss. Natural images, Gaussian mixtures and dead leaves. In *Advances in Neural Information Processing Systems 25*, pages 1745–1753. Curran Associates, Inc., 2012.