Paper Review

MADE: Masked Autoencoder for Distribution Estimation

Philipp Grafendorfer
Rene Maier
Markus Steinmaßl
University of Salzburg
A-5020 Salzburg

Abstract

In this document we provide a review of **MADE: Masked Autoencoder for Distribution Estimation**. The paper was published by Mathieu Germain, Karol Gregor, Iain Murray and Hugo Larochelle on June 5^{th} 2015. They have proposed two main concepts:

- The decomposition of a joint density distribution into a product of conditionals to obtain an autoregressive structure for artificial neural networks.
- Weight masking with masking matrices to apply the decomposition to a deep autoencoder networks and thus create a generative ensemble model by order agnostic training.

The resulting autoencoder outputs probability densities for every previous input. There is plenty of research in this field and the concept of masking weights matrices has been applied to many other interesting architectures like masked autoregressive flow for density estimation by George Papamakarios et al. published on June $14th$2018. During the course of Computer Vision we have come across some interesting concepts, metrics and distributions for computer vision applications. A small part of these concepts shall be reviewed here in the context of **MADE**.

Contents

1 Introduction

Density estimation is a significant topic in unsupervised learning. Revealing the density of data is equivalent to getting deep insights into the underlying mechanisms of data generation. In recent years three strong mechanisms of density estimation have emerged.

- GANS (Generative Adversarial Networks). This architecture is very powerful especially in creating images.
- Variational Autoencoders for variational inference.
- Autoregressive deep neural networks

They all belong to the family of unsupervised learning models. Density estimators can be used for probabilistic inference. If you - for instance - want to infer the other half of an image where only one half is given, this might be a good application. In that case the density of the corrupted half of the image is the posterior density of the non- corrupted half of the image.

The biggest challenge with neural density estimators is the difficulty to ensure that the outputs are actual densities s.t. they satisfy the properties of a density (non-negativity and normalization). To impose restrictions on the architecture in order to achieve that needs effort. In MADE this is achieved by using autoencoder architecture s.t. each output is the parameter of the corresponding input. This is where the properties of autoencoders are expanded. The output parameters are forced in a way so that they maximize the likelihood by minimize the negative log likelihood and therefore make the connection to the binary cross entropy.

2 Underlying concepts

In information theory

the **self information** of an event $\mathrm{x} = x$ is defined as

$$I(x) = \log \frac{1}{P(x)} = -\log P(x) \tag{1}$$

Usually the binary logarithm is used. In this review we stick with base e if not mentioned differently. As the number $\frac{1}{P(x)}$ can be interpreted as the factor with which the probability $P(x)$ has to be multipilied to reach the certainty, the logarithm gives the measure a new unit: **nats** if its the natural logarithm or **bits** if its the logarithm of base 2. The logarithm function is strictly monotonic and therefore suited for a functional transformation like in the self information. One nat is the information gained by observing an event of probability $\frac{1}{e}$. From this function of a random variable x we can obtain the amount of uncertainty in an entire probability distribution by investigating the expected amount of information in an event drawn from that distribution.

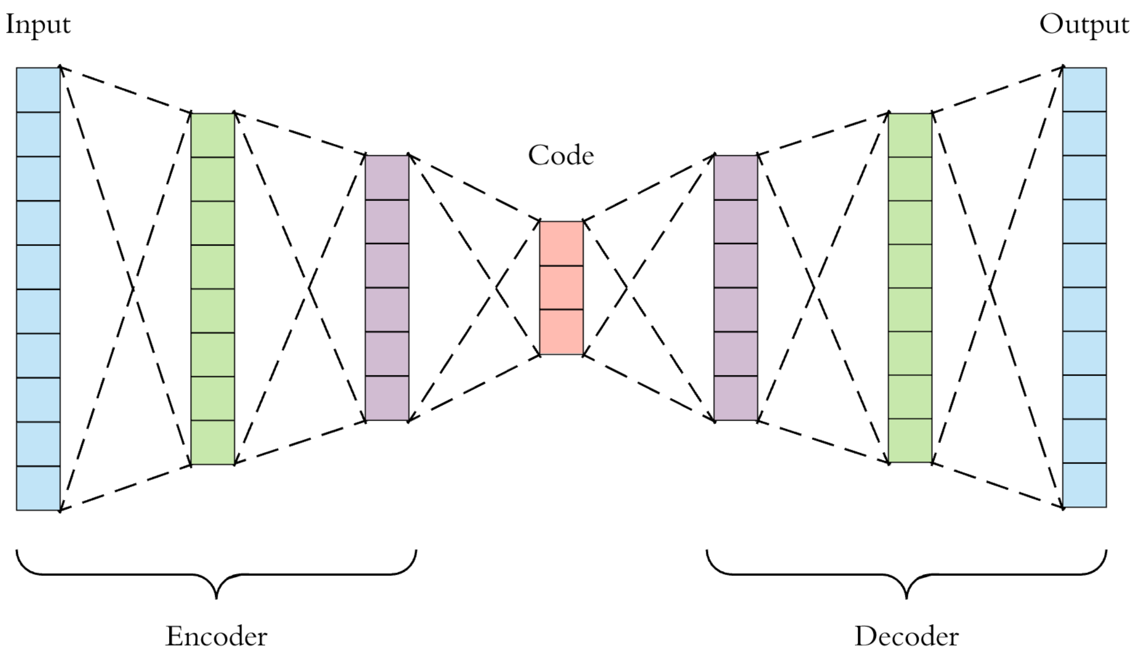


Figure 1: Deep AE
source: Towards Data Science

Entropy and Cross Entropy

The term

$$H(x) = \mathbb{E}_{x \sim P}(I(x)) = -\mathbb{E}_{x \sim P}(\log P(x)) = -\sum_{x \in \mathcal{X}} p(x)\, \log p(x) \tag{2}$$

is called the **Shannon Entropy** of the distribution P.
The cross entropy

$$H(p, q) = -\sum_{x \in \mathcal{X}} p(x)\, \log q(x) \tag{3}$$

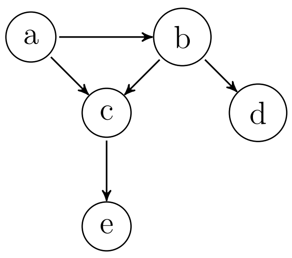
between two probability distributions p and q over the same underlying set of events measures the average number of nats (if the logarithm is based on e) or bits (base 2) needed to identify an event drawn from the set, if a coding scheme is used that is optimized for an "artificial" probability distribution q, rather than the "true" distribution p.[1]

Autoencoder (AE)

An AE maps input to output data. As artificial neural networks are function approximations the AE represents the architecture that learns the identity mapping/function. Figure 1 shows the basic scheme of an AE. There is an encoding and a decoding part.

For more information on the standard AE you might visit the link in the caption of figure 1.

[1] Wikipedia on Cross Entropy



Directed graphical model for random variables
This graph corresponds to probability distributions that can be factored as

$$p(a,b,c,d,e) = p(a)p(b|a)p(c|a,b)p(d|b)p(e|c)$$

Figure 2: Directed Graph

Bernoulli distribution

Given a data set $\{x^{(t)}\}_{t=1}^{T}$ the (joint) **Bernoulli data distribution** given by

$$q(\mathbf{x}) = \prod_{d=1}^{D} \hat{x}_d^{x_d}(1-\hat{x}_d)^{1-x_d}, \tag{4}$$

where the different x_d i.i.d. random variables with $x_d \in \{0,1\}$. Therefore $\hat{x}_d$ is the probability that x_d takes the value 1.

Chain rule of probability

If the input data is not binary it can always be binarized by one-hot encoding the features. So at first sight the binary case might seem special although it is very general. We will derive our loss function in the next section from the Bernoulli distribution by decomposing the joint density (on non iid random variables) into its **conditionals via the chain rule of probability**

$$p(\mathbf{x}) = \prod_{d=1}^{D} p(x_d|\mathbf{x}_{<d}) = \prod_{d=1}^{D} p(x_d|x_1,\dots,x_{i-1}), \tag{5}$$

where $\mathbf{x}_{<d} = [x_1,\dots,x_{d-1}]^T$. A toy example for calculating conditional probabilities is given by the graph in figure 2.

The standard binary cross entropy

$$\sum_{d=1}^{D} -x_d \log \hat{x}_d - (1-x_d)\log(1-\hat{x}_d) \tag{6}$$

cannot be used. The joint density estimator is not allowed to take x_d into consideration for calculating the output $\hat{x}_d$ as the AE could simply learn to map the input to its probability (or more general its parameter) for all input data this might yield a useless density estimator that does not normalize ($\sum_{\mathbf{x}} q(\mathbf{x}) \neq 1$) but just copy data.

3 Autoregression

To satisfy the requirements of a density estimator, the loss functions has to be adapted s.t. it calculates the binary cross entropy for each conditional output $\hat{x}_d$ of the AE.

$$\begin{aligned}
-\log p(\mathbf{x}) &= \sum_{d=1}^{D} -\log p(x_d|\mathbf{x}_{<d}) \\
&= \sum_{d=1}^{D} -\log \hat{x}_d^{x_d}(1-\hat{x}_d)^{1-x_d} \\
&= \sum_{d=1}^{D} -x_d \log p(x_d = 1|\mathbf{x}_{<d}) - (1-x_d)\log p(x_d = 0|\mathbf{x}_{<d}) \\
&= l(\mathbf{x})
\end{aligned} \tag{7}$$

Now the AE can be used for distribution estimation as it forms a proper distribution if each output unit $\hat{x}_d$ only depends on the previous input units. This is referred to as the **autoregressive property**. Computing the negative log- likelihood from equation 7 is equivalent to sequentially predicting each dimension of input $\mathbf{x}$.

4 Masking

The second clever idea is how this autoregressive property is utilized in MADE. The authors decided to define masking matrices which allow it to simply connect output units only to those input units whose dimension index is lower so to compute the loss from equation 7 exactly the way it is meant to be. The construction rules for that matrices are not explained explicitly here. It is quite easy and is done by uniform sampling the number of possible connections of each hidden layer. An example of a result is given in 3. The numbers in the hidden units are those sampled numbers.

For a single hidden layer AE this means

$$\mathbf{h}(\mathbf{x}) = \mathbf{g}(\mathbf{b} + (\mathbf{W} \odot \mathbf{M^W})\mathbf{x} \tag{8}$$

$$\hat{\mathbf{x}} = \text{sigm}(\mathbf{x} + (\mathbf{V} \odot \mathbf{M^V})\mathbf{h}(\mathbf{x})) \tag{9}$$

It is shown that if you calculate the matrix product between input and output masking matrix $\mathbf{M^{V,W}} = \mathbf{M^V M^W}$ the resulting matrix has to be a strictly lower diagonal matrix; only in that case the AE has the necessary autoregressive property. This idea can be expanded to deeper networks easily as the example in figure 3 shows.

For deeper networks the resulting product masking matrix must be strictly lower diagonal.

$$M^{W^1} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, M^{W^2} = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}, M^V = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

From right to left multiplication yields

$$M^V \odot M^{W^2} \odot M^{W^1} = \begin{pmatrix} 0 & 0 & 0 \\ 6 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

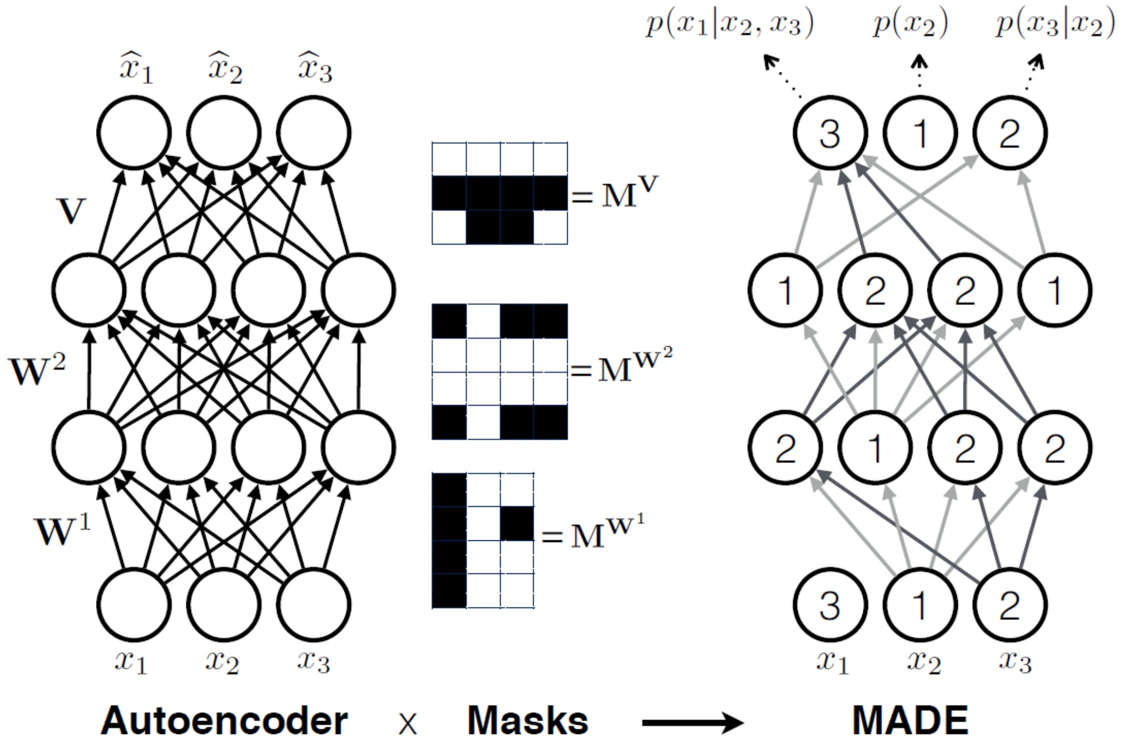
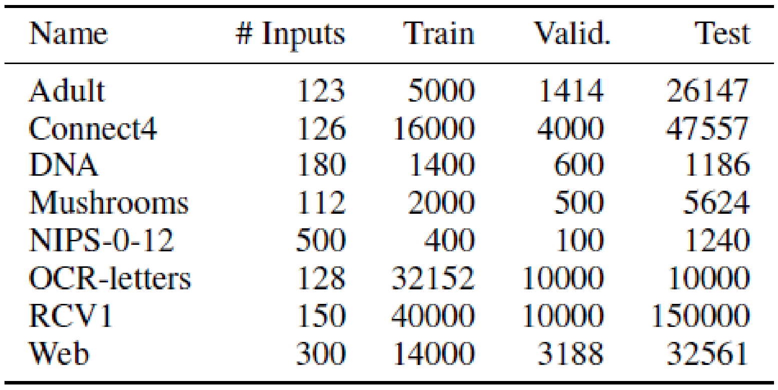
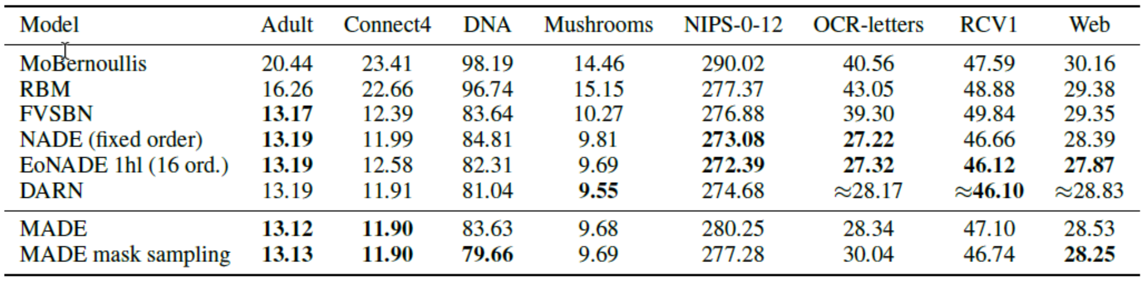


Figure 3: **Left: Conventional three hidden layer AE.** Input in the bottom is passed through fully connected layers and point-wise nonlinearities. In the final top layer, a reconstruction specified as a probability distribution over inputs is produced. As this distribution depends on the input itself, a standard AE cannot predict or sample new data. **Right: MADE.** The network has the same structure as the AE, but a set of connections is removed such that each input unit is only predicted from the previous ones, using multiplicative binary masks (M^{W^1}, M^{W^2}, M^V). In this example, the ordering of the input is changed from 1,2,3 to 3,1,2. The numbers in the hidden units indicate the maximum number of inputs on which the k^{th} unit of layer l depends. The masks are constructed based on these numbers (see Equations 12 and 13). These masks ensure that MADE satisfies the autoregressive property, allowing it to form a probabilistic model, in this example $p(x) = p(x_2)p(x_3|x_2)p(x_1|x_2,x_3)$. Connections in light gray correspond to paths that depend only on 1 input, while the dark gray connections depend on 2 inputs.



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

Figure 4: Scale of evaluation datasets



Model	Adult	Connect4	DNA	Mushrooms	NIPS-0-12	OCR-letters	RCV1	Web
MoBernoullis	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
NADE (fixed order)	**13.19**	11.99	84.81	9.81	**273.08**	**27.22**	46.66	28.39
EoNADE 1hl (16 ord.)	**13.19**	12.58	82.31	9.69	**272.39**	**27.32**	**46.12**	**27.87**
DARN	13.19	11.91	81.04	**9.55**	274.68	≈28.17	≈**46.10**	≈28.83
MADE	**13.12**	**11.90**	83.63	9.68	280.25	28.34	47.10	28.53
MADE mask sampling	**13.13**	**11.90**	**79.66**	9.69	277.28	30.04	46.74	**28.25**

Figure 5: Evaluation and Comparison with other techniques

The resulting is strictly diagonal and the masking matrices therefore fulfill the autoregressive property. In figure 3 the order of the input was changed. Uria et al. (2014) have shown that training an autoregressive model on all orderings of input data can be beneficial. This property is called order agnostic. It can be achieved by sampling the order before each stochastic/minibatch gradient update of the model. One advantage of order agnostic training is straightforward imputation of training data. For missing input data you chosse an ordering s. t. all non-missing inputs are come before unobserved one. This form of training allows parallel generation of an ensemble model where you can average your inference over those different ordering models and therefore make the inference more robust according to its underlying probability distribution.

5 Results

Negative log-likelihood evaluation shows different results on multiple datasets. On some datasets MADE performs best. Overall its performance seems quite reasonable compared to the simplicity of its architecture. The number of input dimensions and examples in training, test and validation set can be found in figure 4 and the performance comparison of the different models on different datasets in figure 5.