Structured Neural Networks for Density Estimation

Asic Q Chen [1] **Ruian Shi** [1] **Xiang Gao** [1] **Ricardo Baptista** [2] **Rahul G Krishnan** [1]

Abstract

Given prior knowledge on the conditional independence structure of observed variables, often in the form of Bayesian networks or directed acyclic graphs, it is beneficial to encode such structure into neural networks during learning. This is particularly advantageous in tasks such as density estimation and generative modelling when data is scarce. We propose the Structured Neural Network (StrNN), which masks specific pathways in a neural network. We design the masks by exploring a novel relationship between neural network architectures and binary matrix factorization, ensuring the desired conditional independencies are respected and predefined objectives are explicitly optimized. We devise and study practical algorithms for this otherwise NP-hard problem. We demonstrate the utility of StrNN in by applying StrNN to binary and Gaussian density estimation tasks. Our work opens up new avenues for applications such as data-efficient generative modeling with autoregressive flows and causal inference.

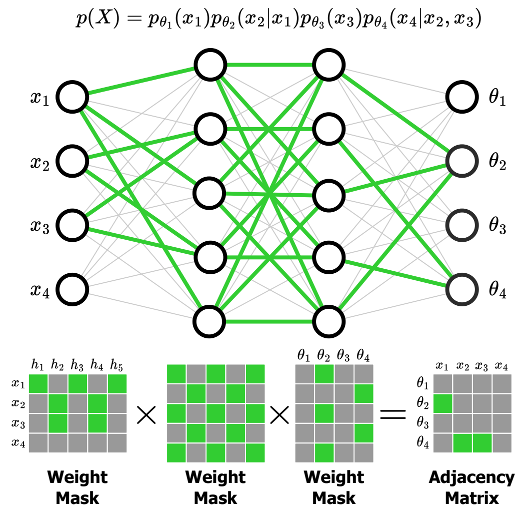


Figure 1. StrNN injects structure by masking the weights of a neural network. *Top*: StrNN connections (green) compared to a fully connected network (gray). *Bottom*: Binary factorization of an adjacency matrix yields weight masks. Masked weights are shown in gray. We use θ to denote network outputs representing the parameters of the marginal conditional distributions for X, and h to represent hidden units.

1. Introduction

The incorporation of (inferred or known) structure into a machine learning model can mitigate the learning of spurious associations, and in turn provide benefits for model generalization, learning efficiency, and interpretability. The improvements are particularly salient when learning from small amounts of data. This idea has found use in reinforcement learning (Ok et al., 2018), computational healthcare (Cui et al., 2020), time series analysis (Curi et al., 2020), causal inference (Balazadeh et al., 2022), and satellite data processing (Katzfuss et al., 2020).

This work focuses on the problem of density estimation of high-dimensional data which has been approached through a variety of lenses. Latent variable models like variational autoencoders (Kingma & Welling, 2013) model data as the marginal over an unobserved latent variable. Normalizing flows (Tabak & Turner, 2013; Rezende & Mohamed, 2015) model data by transforming a base distribution through a series of invertible transformations. Masked autoencoders (MADE) (Germain et al., 2015) model the joint distribution via an autoregressive factorization of the random variables. The factorization is enforced by integrating structure in the neural network of the autoencoder. Specifically, the MADE architecture zeros out weights in a neural network such that each output dimension has an autoregressive dependence of the input dimensions. In practice, this may lead to over-fitting and harm generalization. When knowledge of a Bayesian network and the associated conditional independencies exist, it is desirable to inject this knowledge directly into the network to improve density estimation. In this work, we extend the concept of weight masking, as in MADEs, beyond autoregressive dependencies of the output on the input.

[1]Department of Computer Science, University of Toronto, Toronto, Ontario, Canada [2]Department of Computing & Mathematical Sciences, California Institute of Technology, Pasadena, California, USA. Correspondence to: Asic Chen <asic.chen@mail.utoronto.ca>.

Accepted to the ICML 2023 workshop on Structured Probabilistic Inference & Generative Modeling, Honolulu, Hawaii, USA.

We propose the **Structured Neural Network (StrNN)**, a weight masked neural network capable of enforcing prescribed conditional independencies between variables; see Figure 1. Any set of conditional independencies (e.g. in a Bayesian network) may be represented via an binary adjacency matrix. StrNN performs binary matrix factorization to generate a set of weight masks that respect a given independence structure. There are two key challenges we overcome. First, the general problem of binary matrix factorization in this context is under-specified i.e. there exist many valid masks whose matrix product realizes a given adjacency matrix. To this end, we propose the idea of path maximization as a strategy to guide the generation of valid masks. Second, binary matrix factorization is NP-hard in general. Hence, we propose and study practical solutions that generate valid mask matrices.

Our main contribution is StrNN, a weight-masked neural network which allows injection of prior domain knowledge in the form of Bayesian networks. We formalize the weight masking as an optimization problem, where we can pick the objective based on desired neural architectures. We propose an efficient binary matrix factorization algorithm to mask neural networks of arbitrary sizes. In Appendix B, we introduce a natural application of StrNN: leveraging the structured neural network as the conditioner in an autoregressive normalizing flow, Structured Autoregressive Flows (StrAF) performs well in density estimation and sample generation tasks.

2. Background

Masked Autoencoders for Density Estimation (MADE): Masked neural networks were introduced for density estimation on binary-valued data (Germain et al., 2015). Given $\mathbf{x} = (x_1, ..., x_d)$, MADE factorizes $p(\mathbf{x})$ as the product of the outputs of a neural network. Writing the j-th output as the conditional probability $\hat{x}_j := p(x_j = 1|\mathbf{x}_{<j})$, the joint distribution can be rewritten exactly as the binary cross-entropy loss. As long as the neural network outputs are autoregressive in relation to its inputs, we can minimize the cross-entropy loss for density estimation. To enforce the autoregressive property for a neural network $y = f(x)$ with a single hidden layer and d-dimensional inputs and outputs, MADE multiplies the weight matrices element-wise with binary masks M^W and M^V in the hidden layer as follows:

$$\begin{aligned} h(x) &= g((W \odot M^W)x + b) \\ y &= f((V \odot M^V)h(x) + c) \end{aligned} \tag{1}$$

The autoregressive property is satisfied as long as the product of the masks, $M^V M^W \in \mathbb{R}^{d\times d}$, is lower triangular. The MADE masking algorithm (2) can be extended to neural networks with an arbitrary number of hidden layers and hidden sizes. For Gaussian data, the MADE model can be extended as $\mathbb{R}^d \to \mathbb{R}^{2d}$, $(x_1, ...x_d) \to (\hat{\mu}_1, ..., \hat{\mu}_d, \log(\hat{\sigma}_1), ..., \log(\hat{\sigma}_d))$. The mask between last two layers of the neural network must be duplicated to ensure μ_j and σ_j only depend on $\mathbf{x}_{<j}$. MADE can also be used as the conditioner in an autoregressive flow to model general data, as seen in Papamakarios et al. (2017).

3. Methodology: Sturctured Neural Networks (strNN)

For data $\mathbf{x} = (x_1, ..., x_d)$, a lower-triangular adjacency matrix $A \in \{0, 1\}^{d\times d}$ represents the underlying dependence structure of the individual elements. In other words, $A_{ij} = 0$ for $j < i$ if and only if $x_i \perp x_j | x_{\{1,...,i\}\setminus j}$ and $A_{ij} = 1$ otherwise. This matrix encodes the same information as a Bayesian network DAG of the variables. In the fully autoregressive case, matrix A is a dense lower triangular matrix with all ones under the diagonal, which is the only case the authors of (Germain et al., 2015) addressed. Their proposed MADE algorithm (Appendix A.1) only encodes the structure of dense adjacency matrices into neural networks, and cannot incorporate additional conditional independencies between variables.

We extend the idea of masked autoregressive neural networks to directly encode the independence structure represented by an adjacency matrix A that is lower triangular but also has added sparsity. We observe that a masked neural network satisfies the structural constraints prescribed in A if the product of the masks for each hidden layer has the same locations of zero and non-zero entries as A. Therefore, given the conditional independence structure of the underlying data generating process, we can encode structure into an autoregressive neural network by *factoring the adjacency matrix into binary mask matrices for each hidden layer*.

More specifically, given an adjacency matrix $A \in \{0, 1\}^{d\times d}$ and a neural network with L hidden layers, each with $h_1, h_2, ..., h_L$ hidden units ($\geq d$), we seek mask matrices $M^1 \in \{0, 1\}^{h_1\times d}, M^2 \in \{0, 1\}^{h_2\times h_1}, \ldots, M^L \in \{0, 1\}^{d\times h_L}$ such that $A' \sim A$, where $A' := M^L \cdots M^2 \cdot M^1$. We use $A' \sim A$ to denote that matrices A' and A share the same sparsity pattern, i.e., both matrices have the same locations of zeros and non-zeros. Note that here A is a binary adjacency matrix, and A' is an integer-valued matrix. We then mask the neural network's hidden layers using $M^1, M^2, ..., M^L$ as per (1) to obtain a **Structured Neural Network (StrNN)**, which respects the prescribed independence constraints.

Finding a solution to this problem itself is NP-hard since binary matrix factorization can be reduced to the biclique covering problem (Miettinen & Neumann, 2020; Ravanbakhsh et al., 2016; Orlin, 1977). Furthermore, most existing works focus on deconstructing a given matrix A into low-rank factors while minimizing (but not eliminating) reconstruction error (Dan et al., 2015; Fomin et al., 2020). In our application, we should not allow any reconstruction error since it would break the independence structure we want to enforce in our masked neural network. Hence, existing efficient algorithms for approximate binary matrix factorization are outside the scope of our paper. We instead consider the problem of finding factors that reproduce the adjacency matrix exactly.

We note that the sparsity of the adjacency matrix A satisfying the conditional independence properties of $\mathbf{x}$ will in general depend on the ordering of the variables. In this work, we do not address the problem of seeking a variable ordering that yields the sparsest adjacency matrix A and resulting masks. We refer the reader to (Cundy et al., 2021) for a recent contributions in this direction within the context of learning directed acyclic graphs.

Optimization Objectives. Identifiability of the masks remains an issue even when we eliminate reconstruction error. Given an adjacency matrix A, there can be multiple solutions for factoring A into per-layer masks that satisfy the constraints, especially if the dimensions of the hidden layers are much larger than d. Since the masks dictate which connections remain in the neural network, the chosen mask factorization algorithm directly impacts the neural network architecture. Hence, it is natural to explicitly specify a relevant objective to the neural network's approximation error during the matrix factorization step, such as the test log-likelihood.

Given that the approximation error is inaccessible when selecting the architecture, we were inspired by the Lottery Ticket Hypothesis (Frankle & Carbin, 2018) and other pruning strategies (Srivastava et al., 2014; Gal & Ghahramani, 2016) that identify a subset of valuable model connections. Our hypothesis is that given the same data and prior knowledge on independence structure, the masked neural network with more connections is more expressive, and will thus be able to learn the data better and/or more quickly. To find such models, we consider two objectives: Equation (2) that maximizes the number of input/outputs connections in the neural network while respecting the conditional independence statements dictated by the adjacency matrix, and Equation (3) that maximizes connections while penalizing any pair of variables from having too many connections at the cost of the others. That is,

$$\max_{A' \sim A} \sum_{i=1}^{d} \sum_{j=1}^{d} A'_{ij} \tag{2}$$

$$\max_{A' \sim A} \sum_{i=1}^{d} \sum_{j=1}^{i} A'_{ij} - \text{var}(A') \tag{3}$$

where var(A') is the variance across all entries in A'. While we focus on these two objectives, future work will find optimal architectures by identifying other objectives to improve approximation error.

Factorization Algorithms. Having defined the desired objectives for the mask factorization problem, we consider both exact and approximate algorithms to solve it. We can solve for all the masks jointly, or recursively layer by layer, with some sacrifice on the overall objective. For efficiency, we choose to apply the factorization algorithm one layer at a time. For each objective, we can obtain per-layer exact solutions with integer programming algorithms. While the Gurobi optimizer (Gurobi Optimization, LLC, 2023) can be used for small d, this approach was found to be too computationally expensive for d greater than 20, which is a severe limitation for real-world datasets and models. We hereby propose a greedy algorithm (shown in **Algorithm 1**) that approximates the solution to the maximum connections objective in Equation (2). For each layer, the algorithm first replicates the structure given in the adjacency matrix A by copying its rows into the first mask. It then maximizes the number of neural network connections by filling in the second mask with as many ones as possible while respecting the sparsity in A. See Appendix A.2 for a visual explanation of the algorithm. For a network with d-dimensional inputs and outputs and one hidden layer with h units, this algorithm runs in $\mathcal{O}(dh)$ time. If each of the L hidden layers contains $\mathcal{O}(d)$ units, the overall runtime is $\mathcal{O}(d^2L)$, which is much more efficient than the integer programming solutions. From our experiments, the greedy algorithm executes nearly instantaneously for dimensions in the thousands.

Algorithm 1 Greedy Mask Factorization

```
**Input:** $A \in \{0,1\}^{d_1 \times d_2}$, hidden size $h$.
**Output:** $M^V \in \{0,1\}^{d_2 \times h}$, $M^W \in \{0,1\}^{h \times d_1}$. $nz \leftarrow$ non-zero rows in $A$
Fill $M^W$ with $nz$; repeat until full
Fill $M^V$ with ones
for $i$-th row in $M^V$ do
    $C \leftarrow$ indices of 0's in $i$-th row of$A$
    $T \leftarrow$ cols. of $M^W$ whose index $\in C$
    $R \leftarrow$ indices of non-zero rows of $T$
end for
for $r$ in $R$ do
    Set $M^V_{i,r}$ to zero
end for
```

In Appendix A, we include detailed results from investigating the link between the neural network's generalization performance and the choice of mask factorization algorithm. We observe that while the exact solution to objective (2) achieves a higher objective value than the greedy approach, it has no clear advantage in density estimation performance. Moreover, we found that models trained with the two objectives, (2) and (3), provide similar performance. However, some datasets might be more sensitive to the exact objective. For example, problems with anisotropic non-Gaussian structure may require neural architectures with more expressivity in some variables that may be favored with certain objectives. While we adopted objective (2) and the efficient greedy algorithm in the remainder of our experiments, designing and comparing different factorization objectives is an important direction for future work.

4. Experiments

To demonstrate the efficacy of encoding structure into the learning process, we show that using StrNN to enforce a prescribed adjacency structure improves performance on density estimation tasks. Comparing to the fully autoregressive MADE baseline, we experiment on both synthetic data generated from known structure equations and MNIST image data. Details on data generation for all synthetic experiments can be found in Appendix C.

4.1. Density Estimation on Binary Data

Synthetic Tabular Data We generate binary tabular data through structural equations from known Bayesian networks. The results are shown in Figure 2 (left). We find that StrNN performs better than MADE, especially in the low data regime, as demonstrated on the left hand side of each chart.

MNIST Image Data To study the effect of incorporating structure when modeling images, we use the binarized MNIST dataset considered in (Germain et al., 2015; Salakhutdinov & Murray, 2008). (Germain et al., 2015) treated each 28-by-28-pixel image as a 784-dimensional data vector with full autoregressive dependence. This is a domain for which we do not know the ground truth structure. For StrNN we rely on a local autoregressive dependence on a square of a pixels determined by the hyperparameter `nbr_size`. By changing the hyperparameter we can increase the context window used to model each pixel. StrNN is equivalent to MADE for this experiment when we set `nbr_size=28`. We first find the best `nbr_size` for each label via grid search. For labels 0 and 2, the optimal `nbr_size` is 10, and per-label density estimation results can be found in Figure 2 (right). StrNN outperforms MADE for both labels, more significantly when sample size is small. Samples of handwritten digits generated from both StrNN and MADE can be found in Appendix C.3.

4.2. Density Estimation on Gaussian Data

These experiments compare the performances of StrNN and MADE on synthetic Gaussian data generated from known structure equation models where each x_i is strictly Gaussian. We plot the results in Figure 3. StrNN achieves lower test loss than MADE on average, although the error bars are not necessarily disjoint. When the sample size is low, however, StrNN significantly outperforms MADE, similar to the binary case. In conclusion, across all binary and Gaussian experiments, *encoding structure* makes StrNN significantly more accurate at density estimation than the fully autoregressive MADE baseline.

5. Conclusion & Future Work

In this work, we introduce StrNN, which allows neural networks to encode structure in the form of Bayesian networks using weight masking. For density estimation tasks where the true structure is known, we show that StrNN outperforms the fully autoregressive MADE model on synthetic and MNIST data. The mask factorization algorithm used by StrNN can incorporate different objectives in addition to ensuring the matrices satisfy a sparsity constraint. In section 3, we proposed two such objectives and in Appendix A we demonstrated that they can impact model generalization. StrNN provides a framework with which it is possible to explore other objectives to impose desirable properties on neural network architectures. Investigating the effect of sparse structure on faster and easier training is a valuable future direction (Frankle & Carbin, 2018). Moreover, a natural step is to demonstrate the utility of our framework with structured networks for other applications such as causal effect estimation and variational inference.

References

Balazadeh, V., Syrgkanis, V., and Krishnan, R. G. Partial identification of treatment effects with implicit generative models. *arXiv preprint arXiv:2210.08139*, 2022.

Balgi, S., Peña, J. M., and Daoud, A. Counterfactual analysis of the impact of the imf program on child poverty in the global-south region using causal-graphical normalizing flows. *arXiv preprint arXiv:2202.09391*, 2022a.

Balgi, S., Pena, J. M., and Daoud, A. Personalized public policy analysis in social sciences using causal-graphical normalizing flows. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 11810–11818, 2022b.

Brouillard, P., Lachapelle, S., Lacoste, A., Lacoste-Julien, S., and Drouin, A. Differentiable causal discovery from interventional data. *Advances in Neural Information Processing Systems*, 33:21865–21877, 2020.

Cui, L., Seo, H., Tabar, M., Ma, F., Wang, S., and Lee, D. Deterrent: Knowledge guided graph attention network for detecting healthcare misinformation. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 492–502, 2020.

Cundy, C., Grover, A., and Ermon, S. Bcd nets: Scalable variational approaches for bayesian causal discovery. *Ad-*

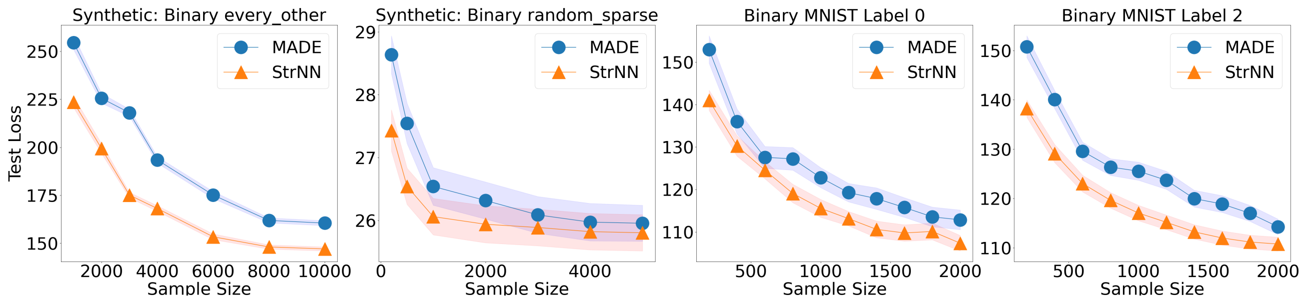


Figure 2. Negative log likelihood of a test set (lower is better) for density estimation experiments on binary synthetic data (left 2 images) and label-dependent binary MNIST data (right 2 images). Error ranges are reported as standard deviation across the test set. StrNN performs better than MADE as the sample size decreases.

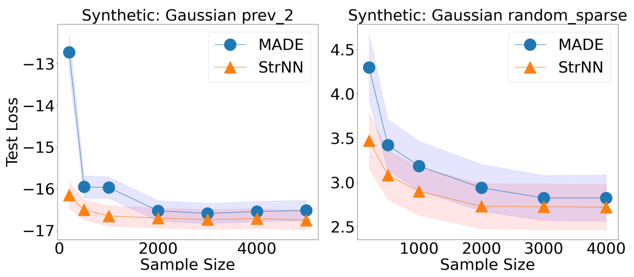


Figure 3. Results from density estimation experiments on Gaussian synthetic data generated from two different sparsity patterns. Test loss is reported in negative log likelihood with error ranges (standard deviation across test set). StrNN performs significantly better than MADE in the low data regime, and better on average.

vances in Neural Information Processing Systems, 34: 7095–7110, 2021.

Curi, S., Melchior, S., Berkenkamp, F., and Krause, A. Structured variational inference in partially observable unstable gaussian process state space models. In *Learning for Dynamics and Control*, pp. 147–157. PMLR, 2020.

Dan, C., Hansen, K. A., Jiang, H., Wang, L., and Zhou, Y. Low rank approximation of binary matrices: Column subset selection and generalizations. *arXiv preprint arXiv:1511.01699*, 2015.

Ding, W., Lin, H., Li, B., and Zhao, D. Causalaf: Causal autoregressive flow for safety-critical driving scenario generation. In *Conference on Robot Learning*, pp. 812–823. PMLR, 2023.

Fomin, F. V., Golovach, P. A., and Panolan, F. Parameterized low-rank binary matrix approximation. *Data Mining and Knowledge Discovery*, 34:478–532, 2020.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Gal, Y. and Ghahramani, Z. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In Balcan, M. F. and Weinberger, K. Q. (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1050–1059, New York, New York, USA, 20–22 Jun 2016. PMLR. URL https://proceedings.mlr.press/v48/gal16.html.

Germain, M., Gregor, K., Murray, I., and Larochelle, H. MADE: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pp. 881–889. PMLR, 2015.

Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Smola, A. A kernel two-sample test. *Journal of Machine Learning Research*, 13(25):723–773, 2012. URL http://jmlr.org/papers/v13/gretton12a.html.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL https://www.gurobi.com.

Huang, C.-W., Krueger, D., Lacoste, A., and Courville, A. Neural autoregressive flows. In *International Conference on Machine Learning*, pp. 2078–2087. PMLR, 2018.

Ilse, M., Forré, P., Welling, M., and Mooij, J. M. Combining interventional and observational data using causal reductions. *arXiv preprint arXiv:2103.04786*, 2021.

Katzfuss, M., Guinness, J., Gong, W., and Zilber, D. Vecchia approximations of gaussian-process predictions. *Journal of Agricultural, Biological and Environmental Statistics*, 25:383–414, 2020.

Khemakhem, I., Monti, R., Leech, R., and Hyvarinen, A. Causal autoregressive flows. In *International conference* *on artificial intelligence and statistics*, pp. 3520–3528. PMLR, 2021.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Marzouk, Y., Moselhy, T., Parno, M., and Spantini, A. An introduction to sampling via measure transport. *arXiv preprint arXiv:1602.05023*, 2016.

Melnychuk, V., Frauen, D., and Feuerriegel, S. Normalizing flows for interventional density estimation. *arXiv preprint arXiv:2209.06203*, 2022.

Miettinen, P. and Neumann, S. Recent developments in boolean matrix factorization. *arXiv preprint arXiv:2012.03127*, 2020.

Mouton, J. and Kroon, S. Graphical residual flows. *arXiv preprint arXiv:2204.11846*, 2022.

Ok, J., Proutiere, A., and Tranos, D. Exploration in structured reinforcement learning. *Advances in Neural Information Processing Systems*, 31, 2018.

Orlin, J. Contentment in graph theory: Covering graphs with cliques. *Indagationes Mathematicae (Proceedings)*, 80(5):406–424, 1977. ISSN 1385-7258. doi: https://doi.org/10.1016/1385-7258(77)90055-5. URL https://www.sciencedirect.com/science/article/pii/1385725877900555.

Papamakarios, G., Pavlakou, T., and Murray, I. Masked autoregressive flow for density estimation. *Advances in neural information processing systems*, 30, 2017.

Papamakarios, G., Nalisnick, E. T., Rezende, D. J., Mohamed, S., and Lakshminarayanan, B. Normalizing flows for probabilistic modeling and inference. *J. Mach. Learn. Res.*, 22(57):1–64, 2021.

Ravanbakhsh, S., Póczos, B., and Greiner, R. Boolean matrix factorization and noisy completion via message passing. In *International Conference on Machine Learning*, pp. 945–954. PMLR, 2016.

Rezende, D. and Mohamed, S. Variational inference with normalizing flows. In *International conference on machine learning*, pp. 1530–1538. PMLR, 2015.

Salakhutdinov, R. and Murray, I. On the quantitative analysis of deep belief networks. In *International Conference on Machine Learning*, 2008.

Silvestri, G., Fertig, E., Moore, D., and Ambrogioni, L. Embedded-model flows: Combining the inductive biases of model-free deep learning and explicit probabilistic modeling. *arXiv preprint arXiv:2110.06021*, 2021.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Tabak, E. G. and Turner, C. V. A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164, 2013.

Wang, R., Chaudhari, P., and Davatzikos, C. Harmonization with flow-based causal inference. In *Medical Image Computing and Computer Assisted Intervention–MICCAI 2021: 24th International Conference, Strasbourg, France, September 27–October 1, 2021, Proceedings, Part III 24*, pp. 181–190. Springer, 2021.

Wehenkel, A. and Louppe, G. Unconstrained monotonic neural networks. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/2a084e55c87b1ebcdaad1f62fdbbac8e-Paper.pdf.

Wehenkel, A. and Louppe, G. Graphical normalizing flows. In *International Conference on Artificial Intelligence and Statistics*, pp. 37–45. PMLR, 2021.

Weilbach, C., Beronov, B., Wood, F., and Harvey, W. Structured conditional continuous normalizing flows for efficient amortized inference in graphical models. In Chiappa, S. and Calandra, R. (eds.), *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pp. 4441–4451. PMLR, 26–28 Aug 2020. URL https://proceedings.mlr.press/v108/weilbach20a.html.

A. Mask Algorithms and Binary Matrix Factorization

A.1. MADE Algorithm and Limitations

As mentioned in the main text, the central idea of masking neural network weights to inject variable dependence was inspired by the work of Germain et al. (Germain et al., 2015). In their paper, the authors proposed Algorithm 2 to ensure the outputs of an autoencoder are autoregressive with respect to its inputs.

Algorithm 2 MADE Masking Algorithm

```
**Input:** Dimension of inputs $d$, Number of hidden layers $L$, Number of hidden units $h$.
**Output:** Masks $M^1, \ldots, M^{L+1}$. $nz \leftarrow$ non-zero rows in $A$
% Sample $\mathbf{m}^l$ vectors
for $l = 1$ to $L$ do
    for $k = 1$ to $h^l$ do
        $\mathbf{m}^l(k) \leftarrow$ Uniform($[\min(\mathbf{m}^{l-1}), \ldots, d-1]$)
    end for
end for
% Construct masks matrices
for $l = 1$ to $L$ do
    $M^l \rightarrow \mathbb{1}_{\mathbf{m}^l \geq \mathbf{m}^{l-1}}$
end for
$M^{L+1} \rightarrow \mathbb{1}_{\mathbf{m}^0 > \mathbf{m}^L}$
```

As a concrete example, let us consider the case of a single hidden layer network with d inputs and h hidden units. Here, Algorithm 2 first defines a permutation $\mathbf{m}^0 \in \mathbb{R}^d$ of the set $\{1, \ldots, d\}$, and then independently samples each entry in the vector $\mathbf{m}^1 \in \mathbb{R}^h$ with replacement from the uniform distribution over the integers from 1 to $d-1$. This assignment is used to define the two binary masks matrices M^1 of size $h \times d$ and M^2 of size $d \times h$. The matrix product of the resulting masks $M^2 M^1 \in \mathbb{R}^{d \times d}$ provides the network's connectivity. In particular, the (i, j) entry of $M^2 M^1$ indicates the dependence of output i on input j.

There are several key limitations of the MADE algorithm:

1. As mentioned in Section 3, the MADE algorithm can only mask neural networks such that they respect the autoregressive property. It is not capable of enforcing additional conditional independence statements as prescribed by an arbitrary Bayesian network. For a general probability distribution that does not satisfy any conditional independence properties, we expect each marginal conditional in the factorization of the density $p(x) = \prod_{k=1}^{d} p(x_k | x_{<k})$ to depend on all previous inputs. As a result, the matrix product $M^2 M^1$ should be fully lower-triangular, meaning that output k depends on all inputs $1, \ldots, k-1$. If there is conditional independence structure, however, the MADE algorithm does not provide a mechanism to define mask matrices such that their matrix product is sparse and hence the corresponding MADE network enforces these constraints on the variable dependence.

2. The non-deterministic MADE masking algorithm presented in (Germain et al., 2015), the resulting mask matrices are not always capable of representing any distribution. In particular, the random algorithm can yield some mask matrices where the lower-triangular part of their matrix product is arbitrarily sparse, i.e., there exists some $k < k'$ such that $(M^2 M^1)_{k,k'} = 0$. As a result, the MADE network with these masks enforces additional conditional independencies that are not necessarily present in the underlying data distribution. Proposition A.1 formalizes this point.

Proposition A.1. *There is a non-zero probability that Algorithm 2 will yield masks that enforce unwanted conditional independencies.*

Proof. For a single hidden layer ($L = 1$) neural network with h units, there is a probability $1/d^h > 0$ of sampling $\mathbf{m}^1 = \mathbf{1}$, i.e., each entry is independently sampled to be 1. This vector yields a mask matrix M^1 that only has one non-zero column of ones at the index k where $\mathbf{m}^0(k) = 1$. As a result, the matrix product $M^2 M^1$ also has only one non-zero column at index k, meaning that all outputs $k' > k$ depend only on x_k and not on other input variables. Therefore, these mask matrices enforce the constraints $X_{k'} \perp\!\!\!\perp X_{<k} | X_k$ for all k'. Equivalently, a distribution that does not satisfy these constraint can not be represented using this MADE network. $\square$

In our work, we adopt a weight masking scheme by solving a binary matrix factorization problem that overcomes these limitations. Both the globally optimal and approximate solutions proposed in our paper are deterministic. Thus, we enforce all conditional independence properties exactly and ensure unwanted variable independence statement do not appear in our neural networks.

A.2. Mask Factorization: Integer Programming and Greedy Algorithm

In Section 3, we showed that finding the weight masks for each neural network layer is equivalent to factoring the adjacency matrix that represents the input and output connectivity of these layers. We can find exact solutions to this problem by solving the optimization problem given in (2). This problem can be formalized as the following integer programming problem:

$$\begin{aligned}
&\text{Inputs: } A \in \{0,1\}^{d\times d} \\
&\text{Outputs: } M^V \in \{0,1\}^{d\times h}, M^W \in \{0,1\}^{h\times d} \\
&\max \sum_{i=1}^{d}\sum_{j=1}^{d} v_i w_j \\
&\text{such that } v_i w_j > 0 \text{ if } A_{ij} = 1 \\
&\qquad v_i w_j = 0 \text{ if } A_{ij} = 0 \\
&\qquad \text{where } M^V = \begin{pmatrix} v_1 \\ v_2 \\ \dots \\ v_d \end{pmatrix} \text{ and } v_i \in \{0,1\}^{1\times h} \\
&\qquad \text{and } M^W = \begin{pmatrix} w_1 & w_2 & \dots & w_d \end{pmatrix} \text{ and } w_j \in \{0,1\}^{h\times 1}
\end{aligned} \tag{4}$$

To formulate a similar problem for the objective given in (3) instead, we simply replace the integer programming objective with

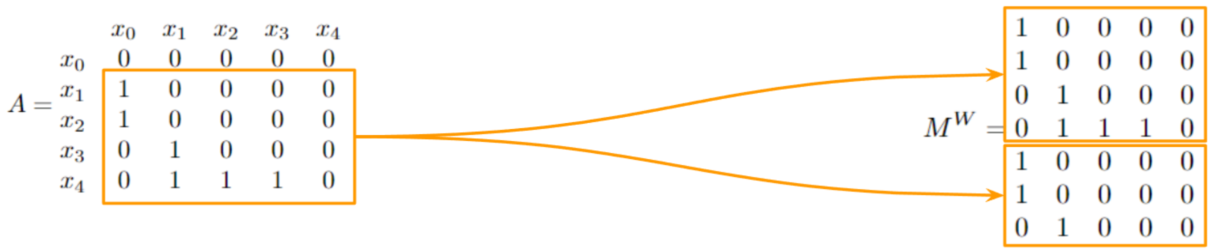
$$\max \Big(\sum_{i=1}^{d}\sum_{j=1}^{d} v_i w_j - \text{Var}_{i,j}(v_i w_j)\Big). \tag{5}$$

We used the Gurobi optimizer (Gurobi Optimization, LLC, 2023) to solve the above integer programming problems in our experiments, and found that exact solutions are found reasonably quickly for up to $d = 20$. For dimensions larger than 20, however, directly solving the integer programming problem becomes prohibitively expensive even on computing clusters with multiple GPUs, so it is intractable to seek exact solutions to these problems for most real-world datasets. Therefore, in this work we propose Algorithm 1, a greedy method that gives an approximate solution to the problem 4 very efficiently. Figure 4 provides a visual example of the steps performed by Algorithm 1.

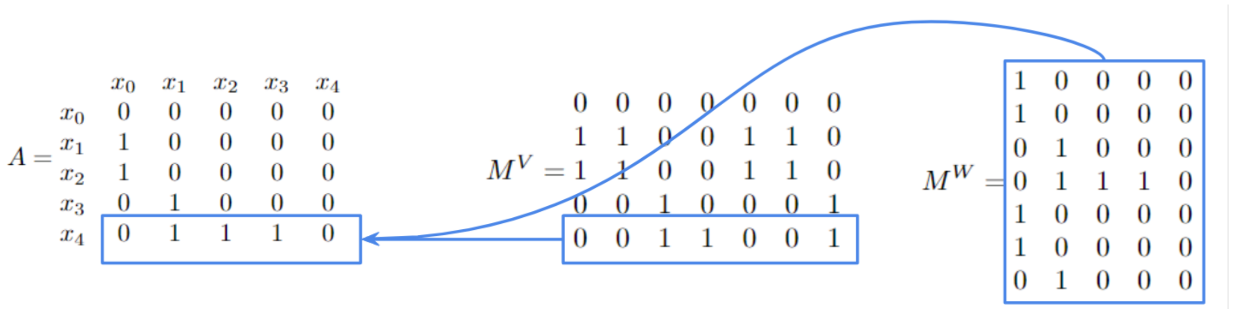
To estimate how well the greedy algorithm approximates the solution to problem 4, we randomly sample lower triangular adjacency matrices, setting entries to 0 or 1 based on a given sparsity threshold between 0 and 1. In other words, for the threshold 0.1, the random adjacency matrix is very dense, and when the threshold is 0.9, it is very sparse. For fixed input and output dimensions, we sample 10 such random adjacency matrices for each sparsity threshold ranging from 0.1 to 0.9, and take the average of the $\sum_{i=1}^{d}\sum_{j=1}^{d} v_i w_j$ objective value obtained by each factorization algorithm. Results for $d = 10$-dimensional adjacency matrices are shown in Figure 5. We see that the exact integer programming solution achieves higher objective values compared to the greedy algorithm we propose in Algorithm 1, but it remains to further evaluate the resulting masks from both algorithms on their performance for a downstream density estimation task.

A.3. Mask Algorithms and Generalization

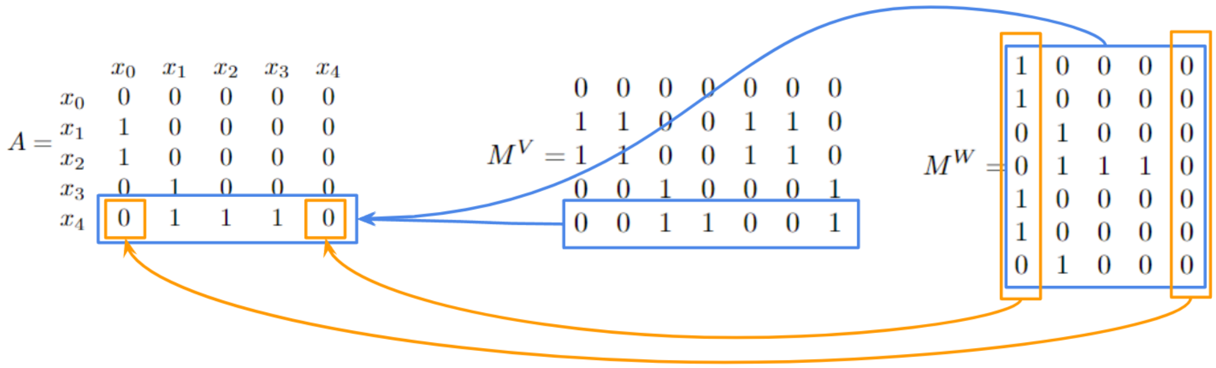
To that end, we evaluate the density estimation performance of masked neural networks on 20-dimensional synthetic binary data, using the following four mask factorization methods:



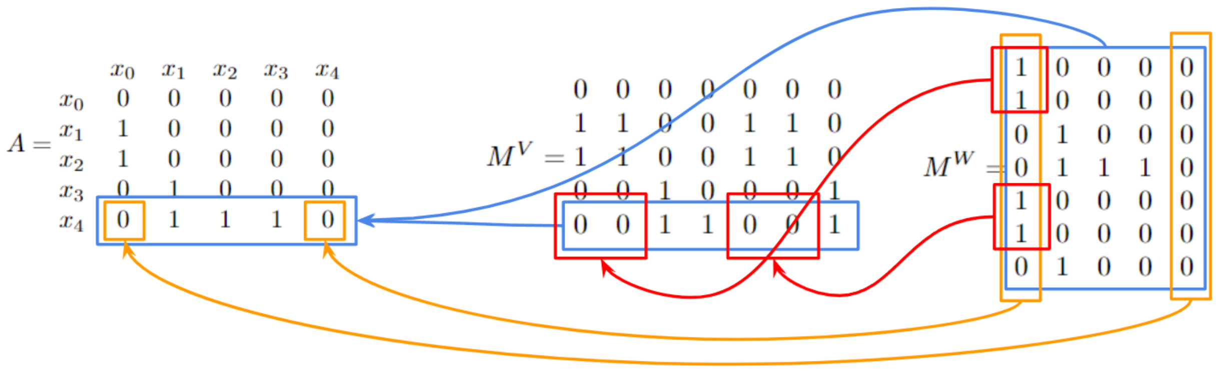
(a) Step 1: Given the adjacency matrix A, we first populate the first layer mask M^W by copying over non-zero rows in A, and repeating until all rows of M^W are full.



(b) Step 2: Next, we populate the second layer mask M^V. Let us take the last row of M^V for an example: to respect all conditional independence statements given by A, we need the product of the last row of M^V and the M^W matrix to have the same zero and non-zero locations as the last row of A. Since there are zeros in the first and last column of A's last row, we need the products of the last row of M^V with the first and last columns of M^W to be zero.

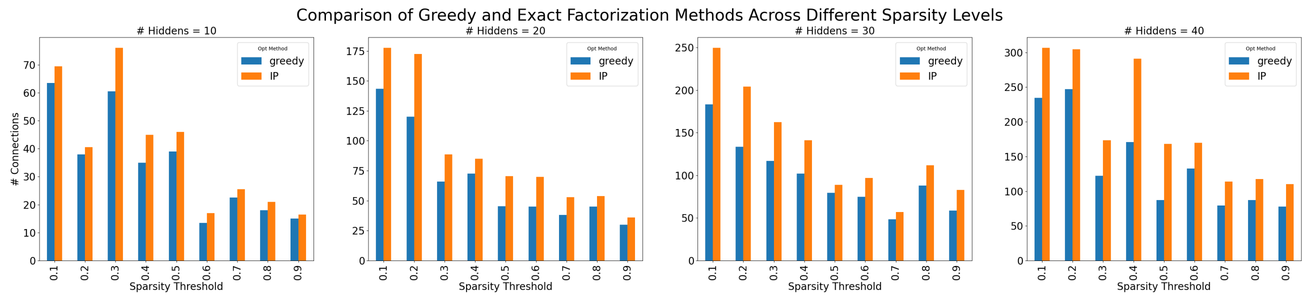


(c) Step 3: We find the unique ones in the first and last columns of M^W and set the corresponding positions in the last row of M^V to zero.

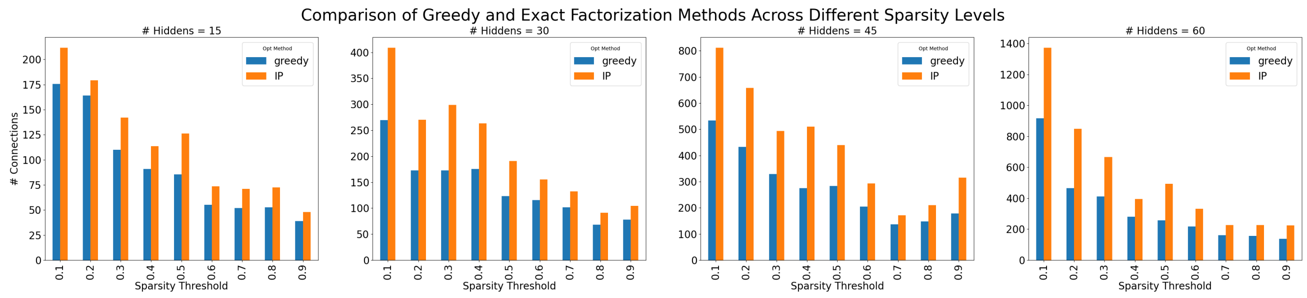


(d) Step 4: Everything else in the last row of M^V is set to 1 to maximize the number of connections in the resulting neural network for the optimization objective in (2).

Figure 4. A visual example of Algorithm 1 being applied on the adjacency matrix A for a neural network with $d = 5$ inputs, $d = 5$ outputs, and one single hidden layer containing $h = 7$ hidden units.



(a) Randomly generated adjacency structures of 10 dimensions.



(b) Randomly generated adjacency structures of 15 dimensions.

Figure 5. Comparing objective value (Equation 2) achieved by greedy and exact integer programming (IP) methods. IP gives better objective values when the adjacency matrix is very sparse. As the number of neurons involved goes up, the difference in these methods also increases.

1. **MADE**: A fully autoregressive baseline using Algorithm 2 as proposed in (Germain et al., 2015)
2. **Greedy**: The proposed method in Algorithm 1
3. **IP**: The exact integer programming solution to Problem 4
4. **IP-var**: The exact integer programming solution to Problem 4, but with the objective in (5)

The experiment setup and grid of hyperparameters used for these experiments are the same as those in all binary and Gaussian experiments in this work, as detailed in Appendix C. Specifically, the adjacency structures used in these experiments are explained and visualized in C.1. The results for the negative log-likelihood of a test dataset are reported in Table 1. We see that all three methods proposed in this work—Greedy, IP, and IP-var—outperform the MADE baseline, but there is no clear winner based on overlapping error ranges. IP does not perform significantly better than Greedy based on the higher objective value achieved for (2). Meanwhile, there is no significant difference between the objective that maximizes the total connections (Equation 2 and the objective with the added variance penalty (Equation 3) when comparing the performance of IP versus IP-var. Hence, for efficiency and overall performance, we choose to adopt the Greedy mask factorization algorithm for the rest of the experiments in this paper.

B. Application: Structured Autoregressive Flows (StrAF)

StrNN can be applied to neural network-based density estimation in various contexts. Where conditional independence properties are known *a priori*, we showed in Section 4 that StrNN can be used to estimate parameters of binary or Gaussian data distributions while keeping specified variables conditionally independent. In order to model real-valued data distributions, we further integrate StrNNs into an autoregressive normalizing flow (Papamakarios et al., 2017; Huang et al., 2018) to form the **Structured Autoregressive Flow (StrAF)**. The StrAF model uses the StrNN as a normalizing flow conditioner network, which respects a given adjacency structure within each flow layer. The StrAF preserves variable orderings between chained layers, allowing the adjacency structure (i.e., the prescribed conditional independence structure by the adjacency matrix) to be respected throughout the entire flow.

Table 1. Density estimation results on 20-dimensional synthetic datasets, reported using the negative log-likelihood on a held-out test set (lower is better). The error reported is the sample variance across the test set. The three methods, Greedy, IP, and IP-var perform better than the MADE baseline, but similarly to each other.

Dataset	Random Sparse		Previous 3	
	$n = 5000$	$n = 2000$	$n = 5000$	$n = 2000$
MADE	7.790 ± 0.140	7.788 ± 0.142	8.767 ± 0.132	8.816 ± 0.134
Greedy	7.758 ± 0.137	7.778 ± 0.142	8.757 ± 0.131	**8.768** $\pm$ **0.130**
IP	7.758 ± 0.138	7.769 ± 0.140	**8.755** $\pm$ **0.132**	8.769 ± 0.129
IP-var	**7.757** $\pm$ **0.137**	**7.768** $\pm$ **0.140**	8.758 ± 0.132	8.770 ± 0.131

Dataset	Every Other	
	$n = 5000$	$n = 2000$
MADE	8.373 ± 0.120	8.364 ± 0.124
Greedy	8.334 ± 0.125	8.315 ± 0.123
IP	8.333 ± 0.129	**8.314** $\pm$ **0.123**
IP-var	**8.331** $\pm$ **0.126**	**8.314** $\pm$ **0.124**

B.1. Background on Autoregressive Flows

Normalizing Flows: Normalizing flows (Rezende & Mohamed, 2015) model complex data distributions and have been applied in many scenarios (Papamakarios et al., 2021). Given data $\mathbf{x} \in \mathbb{R}^d$, a normalizing flow $\mathbf{T}\colon \mathbb{R}^d \to \mathbb{R}^d$ takes $\mathbf{x}$ to latent variables $\mathbf{z} \in \mathbb{R}^d$ that are distributed according to a simple base distribution $p_{\mathbf{z}}$, such as the standard normal. The transformation $\mathbf{T}$ must be a diffeomorphism (i.e., differentiable and invertible) so that we can compute the density of $\mathbf{x}$ via the change-of-variables formula: $p_{\mathbf{x}}(\mathbf{x}) = p_{\mathbf{z}}(\mathbf{T}(\mathbf{x}))|\det J_{\mathbf{T}}(\mathbf{x})|$. We can compose multiple diffeomorphic transformations $\mathbf{T}_k$ to form the flow $\mathbf{T} = \mathbf{T}_1 \circ \cdots \circ \mathbf{T}_K$ since diffeomorphisms are closed under composition. The flows are trained by maximizing the log-likelihood of the observed data under the density $p_{\mathbf{x}}(\mathbf{x})$. The log-likelihood can be evaluated efficiently when it is tractable to compute the Jacobian determinant of $\mathbf{T}$; for example when $\mathbf{T}_k$ is a lower triangular function (Marzouk et al., 2016). Given the map, we can easily generate i.i.d. samples from the learned distribution by sampling from the base distribution $\mathbf{z}^i \sim p_{\mathbf{z}}$ and evaluating the flow $\mathbf{T}^{-1}(\mathbf{z}^i)$.

Density Estimation with Autoregressive Flows: One special case of normalizing flows considers a single layer where the Jacobian matrix is lower triangular, in which case its determinant is simply the product of its diagonal entries (Huang et al., 2018). This gives rise to the autoregressive flow formulation: given an ordering π of the d variables in the data vector $\mathbf{x}$, the jth component of the flow $\mathbf{T}$ has the form: $x_j = \tau_j(z_j; c_j(\mathbf{x}_{<\pi(j)}))$ where each τ_j is an invertible *transformer* and each c_j is a *conditioner* that only depends on the variables that come before x_j in the ordering π. As a result, the map components define an autoregressive model that factors the density over a random variable x as: $p(\mathbf{x}) = \prod_{j=1}^d p(x_j|\mathbf{x}_{<j})$ where $\mathbf{x}_{<j} = (x_1, \ldots, x_{j-1})$.

Under mild conditions, any arbitrary distribution $p_{\mathbf{x}}$ can be transformed into a base distribution with a lower triangular Jacobian matrix (Rezende & Mohamed, 2015). That is, autoregressive flows are arbitrarily expressive given the target distribution. One common choice of invertible functions for the transformer are monotonic neural networks (Wehenkel & Louppe, 2019).

B.2. Related Works on Structured Normalizing Flows

Given a Bayesian network adjacency matrix, Wehenkel & Louppe (2021) introduced graphical conditioners to the autoregressive flows architecture through input masking. They demonstrated that unifying normalizing flows with Bayesian networks showed promise in injecting domain knowledge while promoting interpretability, as even single-step graphical flows yielded competitive results in density estimation. Our work follows the same idea of introducing prior domain knowledge into autoregressive flows, but we instead use a masking scheme similar to methods in Germain et al. (2015).

Silvestri et al. (2021) proposed embedded-model flows, which alternates between traditional normalizing flows layers and gated structured layers that a) encode parent nodes based on the graphical model, and b) include a trainable parameter that determines how strongly the current node depends on its parent nodes, which alleviates error when the assumed graphical model is not entirely correct. In comparison, our work encodes conditional independence more directly in the masking step, improving accuracy when the assumption in the probabilistic graphical structure is strong.

Mouton & Kroon (2022) applied a similar idea to residual flows by masking the residual blocks' weight matrices prior to the spectral normalization step according to the assumed Bayesian network. Similarly, Weilbach et al. (2020) introduced graphical structure to continuous normalizing flows, masking the weight matrices in the neural network used to parameterize the time derivative of the flow. Our work also uses weight masking, but in autoregressive flows. Furthermore, our approach permits explicit optimization of different objectives during the adjacency matrix factorization step. We investigate the efficacy of factorization schemes and resulting neural architectures in our work.

Flows have garnered increasing interest in the context of causal inference, with applications spanning various problem domains. Ilse et al. (2021) parameterized causal model with normalizing flows in the general continuous setting to learn from combined observational and interventional data. Melnychuk et al. (2022) used flows as a parametric method for estimating the density of potential outcomes from observational data. Flows have also been employed in causal discovery (Brouillard et al., 2020; Khemakhem et al., 2021) as well as in various causal applications (Ding et al., 2023; Wang et al., 2021). In particular, Balgi et al. (2022b) also considered embedding the true causal DAG in flows for interventional and counterfactual inference, but they do so via the framework of Graphical Normalizing Flows (Wehenkel & Louppe, 2021). Balgi et al. (2022a) used CAREFL on a real-world social sciences dataset leveraging a theorized Bayesian network.

B.3. Structured Autoregressive Flows Experiments

B.3.1. Synthetic Data Generation

We use randomly generated non-linear and multi-modal dataset with sparse conditional dependencies between variables for autoregressive flow evaluation. The adjacency matrix used is visualized in Figure 6. This adjacency matrix is used to generate masks for StrAF, and is provided to GNF as the ground truth adjacency matrix for its input masking scheme.

The objective is to generate a $d = 15$-dimensional multi-modal and non-linear dataset. We create the ground truth adjacency matrix $A \in \{0, 1\}^{15\times 15}$ by sampling each entry in the matrix independently from the Uniform$(0, 1)$ distribution. Each element is converted to a binary value using a sparsity threshold of 0.8. Moreover, upper triangular elements are then zeroed out. This results in a sparse binary adjacency matrix for which values of one indicate conditional dependence, and zeros indicate conditional independence.

In our data generating process, variables with conditional dependencies are generated as a weighted sum of its preceding dependent variables. We generate a second matrix $W \in \mathbb{R}^{15\times 15}$ containing these weights, where each element is sampled from the Uniform$(-3, 3)$ distribution. This matrix W is then multiplied element-wise by A to zero out pairs of variables that are conditionally independent. If we denote entries of W as w_{ij}, each dependent pair of variables is generated by the following process:

$$x_t = \sqrt{\sum_{j=1}^{t-1} (w_{tj} x_j)^2 + \varepsilon}, \quad \varepsilon \sim \mathcal{N}(0, 1). \tag{6}$$

Variables which are conditionally independent (e.g., x_1) are generated using a mixture of three Gaussians. For each variable and each Gaussian mixture component, we sample its mean from the Uniform$(-8, 8)$ distribution, and its standard deviation from the Uniform$(0.01, 2)$ distribution. We draw the mixture weights from the Dirichlet(1, 1, 1) distribution. At sampling time, we use this mixture weight vector to determine the number of samples to draw from each Gaussian mixture component. For our experiments, we draw 5000 samples using this data generating process, and use a [0.6, 0.2, 0.2] ratio for training / validation / testing splits.

B.3.2. Experimental Results

We evaluate StrAF on density estimation against baselines on the $d = 15$ tri-modal and non-linear synthetic dataset for experimental evaluation. We use 5000 samples and a [0.6, 0.2, 0.2] train / validation / test split.

Experimental Setup We select the fully auto-regressive flow (ARF-10) and the Graphical Normalizing Flow (GNF) (Wehenkel & Louppe, 2021) as the most relevant baselines for comparison. While other structured flows exist and have been examined in Appendix B.2, they do not represent latent variables in an autoregressive structure and are less directly

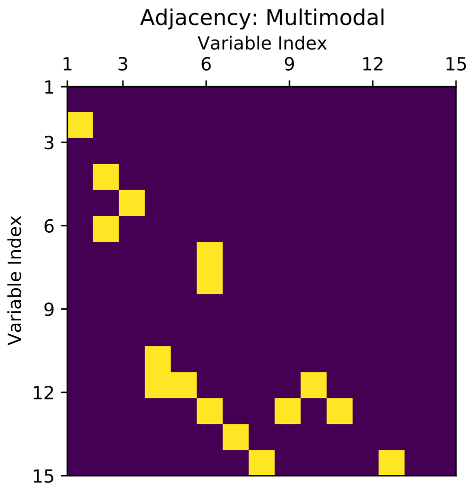


Figure 6. Adjacency matrices used to generate the multi-modal non-linear synthetic dataset used in Section B.3.1. The matrix is also used to generate StrAF masks and for GNF masking. Conditionally dependent variables are shown in yellow.

comparable. Both StrAF and GNF are provided the true adjacency matrix in their conditioners. As GNF permutes variables between flow steps, it only respects a prescribed adjacency when using a single flow step. This limits performance in density estimation, and we highlight this trade-off by comparing single step and ten step models (denoted with -1 and -10). All models use a UMNN (Wehenkel & Louppe, 2019) transformer and we grid-search other hyperparameters in Appendix B.3.3.

We evaluate density estimation performance using the negative log-likelihood (NLL) on test data. We evaluate sample quality by computing the maximum mean discrepancy (MMD) using an RBF kernel ($\gamma = 0.1$) between 640 model generated samples and the ground truth distribution. For each method, we select hyperparameters based on the validation loss, and then train eight runs using random initialization. We then report the mean and 95% CI of each metric using these runs.

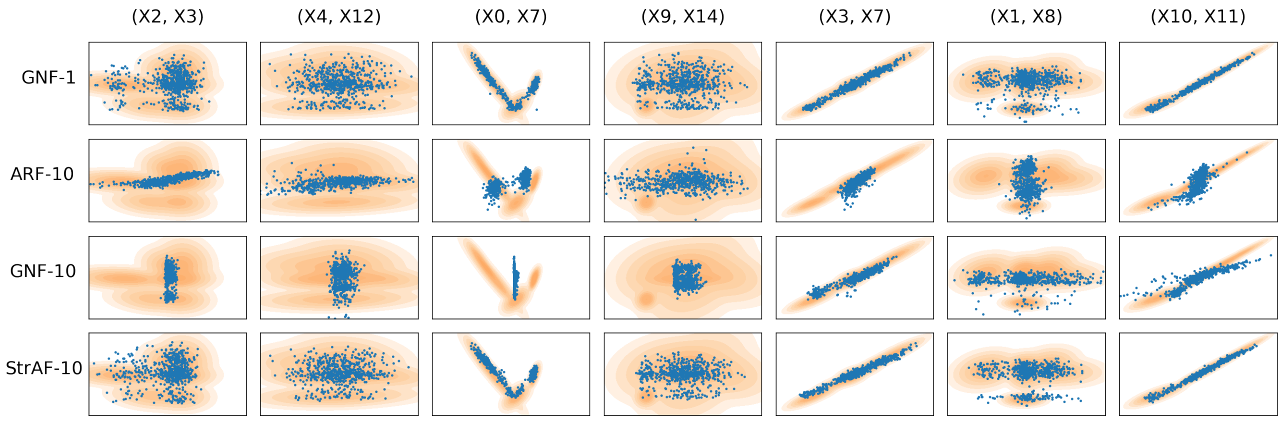


Figure 7. Model generated samples are shown in blue dots for randomly selected dimensions. The ground truth density is visualized by the orange contours. Samples from methods that respect the prescribed structure throughout the entire flow (GNF-1, StrAF-10) match the true distribution better than those that do not.

StrAF improves density estimation without sacrificing sample quality. We report the experimental results in Table 2 and observe several trends. As expected, increasing the number of flow steps improves density estimation. The ability to prescribe the ground truth adjacency structure further increases model generalization in comparison to a fully auto-regressive model. The GNF and ARF permute variables between flow steps, hurting sample quality unless only a single flow step is

Table 2. Evaluation of StrAF versus baselines. Number after method name indicates number of flow steps.

	Test NLL (↓)	MMD (↓)
GNF-1	-3.77 ± 0.18	**0.016 ± 6e-4**
ARF-10	-3.94 ± 0.03	0.177 ± 2e-2
GNF-10	**-4.38 ± 0.03**	0.173 ± 2e-2
StrAF-1	-3.66 ± 0.08	**0.015 ± 5e-4**
StrAF-10	**-4.35 ± 0.01**	0.020 ± 7e-4

used. StrAF does not permute variables between flow steps and is able to estimate the density comparably to GNF-10 while significantly improving sample quality. We visualize model generated samples for several dimensions in Figure 7.

B.3.3. Additional Experiment Details

Method Selection. We compared the StrAF against a fully autoregressive flow (denoted ARF in the main text) and the Graphical Normalizing Flow (GNF) (Wehenkel & Louppe, 2021). The fully autoregressive flow assumes no conditional independencies in the data generation process, hence using a full lower triangular adjacency matrix for masking. Meanwhile, the GNF model also encodes conditional independencies, but cannot be extended past one flow step without sacrificing sample quality, as shown in the main text. Both models were selected as they are autoregressive flows, and we omit comparison to other flows in Appendix B.2 as they are not autoregressive and less relevant for comparison.

Training. Each model is trained using the Adam optimizer for a maximum of 150 epochs using a batch size of 256. During all runs, the models were trained using early stopping on the validation log-likelihood loss with a patience of 10 epochs, after which the model state at the best epoch was selected. We consider two additional training schedules: decreasing the learning rate by a factor of 0.1 on plateaus where the loss does not improve for five epochs (denoted Plateau), and a single scheduled decrease by a factor of 0.1 at epoch 40 (denoted MultiStep). In addition to the standard fixed learning rate, we select between these training schedules as a hyperparameter.

While the GNF can learn an adjacency matrix from data, we are interested in scenarios where an adjacency matrix is prescribed. Thus, we disable the learning functionality of the adjacency in GNF by stopping gradient updates to the GNF input mask matrix. We retain the one hot encoding network described in the GNF paper. The fully autoregressive flow is implemented by using a GNF with a full lower triangular adjacency matrix. Latent variables are permuted in ARF and GNF, as described by their original publications, but we do not permute variables for StrAF.

Hyperparameters. Here we report the process used to select the model hyperparameters. We use the UMNN (Wehenkel & Louppe, 2019) as each flow's transformer. We use 20 integration steps to compute the transformer output. The UMNN is conditioned on values computed by the conditioner. We select the dimension of these values as a hyperparameter named "UMNN Hidden Size". We then determine the best hyperparameters for each method using a grid search with the values in Table 3.

Hyperparameter	Grid Values
Flow Steps	$[1, 5, 10]$
Conditioner Net Width	$[50, 500]$
Conditioner Net Depth	$[3, 4]$
UMNN Hidden Size	$[25, 50]$
UMNN Width	$[250, 500]$
UMNN Depth	$[4, 6]$
Learning Rate	$[0.001, 0.0001]$
LR Scheduler	[Fixed, Plateau, MultiStep]

Table 3. Hyperparameter grid: normalizing flows

Note that while we include the number of flow steps in the grid search, we also reported the results for a single flow step to illustrate the trade off between sample quality and density estimation as a function of flow depth. We found that ten steps always outperformed five steps, and hence we did not report the results for the five step models. The best hyperparameters (as determined by validation loss) that were selected for each model used in the main text are reported in Table 4.

Hyperparameter	ARF-10	GNF-1	GNF-10	StrAF-1	StrAF-10
Flow Steps	10	1	10	1	10
Conditioner Net Width	50	500	50	500	500
Conditioner Net Depth	4	4	3	4	4
UMNN Hidden Size	50	25	50	50	50
UMNN Width	500	500	250	500	250
UMNN Depth	4	4	6	6	6
Learning Rate	0.001	0.001	0.001	0.001	0.001
LR Scheduler	MultiStep	MultiStep	Plateau	Plateau	Plateau

Table 4. Final hyperparameters used per flow model

We fix the best hyperparameters for each model, and then re-train each model on the same data splits using eight new random seeds. This ensemble of eight models was used to generate confidence intervals for the evaluation metrics described below.

Evaluation Metrics. We evaluate each model in two ways: density estimation, and the quality of samples generated. We evaluate the ability for a model to estimate the density underlying the samples by reporting the negative log-likelihood on a held-out test split. Sample quality is evaluated using the maximum mean discrepancy (MMD) (Gretton et al., 2012). We sample from each model by taking a noise sample $\mathbf{z} \sim N(0, I)$ and then transforming it to the data space using the inverted flow ($\mathbf{T}^{-1}$) from each model: $\mathbf{T}^{-1}(\mathbf{z}) = \mathbf{x}$. We repeat this process 640 times, and compare the generated samples against the true data distribution, as represented by 1000 samples from the held-out test split. Denoting the collection of model samples using $\mathbf{X}$ and test split samples as $\mathbf{Y}$, the MMD is computed as:

$$\text{mean}(K(\mathbf{X}, \mathbf{X})) + \text{mean}(K(\mathbf{Y}, \mathbf{Y})) - 2\text{mean}(K(\mathbf{X}, \mathbf{Y})) \tag{7}$$

where $K(\cdot, \cdot)$ denotes the RBF kernel ($\gamma = 0.1$) between two sets of points. When reporting the MMD for each model class, we again use the eight models re-trained using different random seeds, as described in the previous subsection, to report the mean and 95% confidence interval (across re-training runs) of the MMD.

C. Binary & Gaussian Experiments: Data Generation, Experiment Setup, and Additional Results

C.1. Data Generation & Adjacency Structures

For the main StrNN v. MADE experiments in this paper, we consider the following datasets:

1. $d = 800$ synthetic binary dataset where each variable depend on every other preceding variable (”Binary every_other”).
2. $d = 50$ synthetic binary dataset where the adjacency matrix is randomly generated based on sparsity threshold (”Binary random_sparse”).
3. $d = 20$ synthetic Gaussian dataset where each variable is dependent on 2 previous variables (”Gaussian prev_2”).
4. $d = 20$ synthetic Gaussian dataset where the adjacency matrix is randomly generated based on a sparsity threshold (”Gaussian random_sparse”).
5. 784-pixel MNIST handwritten digit images, binarized according to (Salakhutdinov & Murray, 2008).

In Figures 8 and 9, we visualize the adjacency matrices that were used to generate the synthetic datasets listed above.

The true underlying conditional independence structure for the MNIST dataset used in Section 4.1 is unknown, which is also a common challenge for any real-world/image dataset. Instead, when using the StrNN masked neural network, we aim to encode the inductive bias of *locality*, so that density estimation for a single pixel only depends on its surrounding neighbourhood of pixels. For the results shown in the main text, we decided to use a neighbourhood size of 10 after an extensive hyperparameter search for this parameter. As explained briefly in Section 4.1, the hyperparameter `nbr_size` specifies the radius of the square context window originating from each pixel. Each pixel is modelled to be dependent on all previous pixels in that window, as specified by the variable ordering. For variable ordering, we use the default row-major pixel ordering for the MNIST images. The resulting adjacency matrix is visualized in Figure 10.

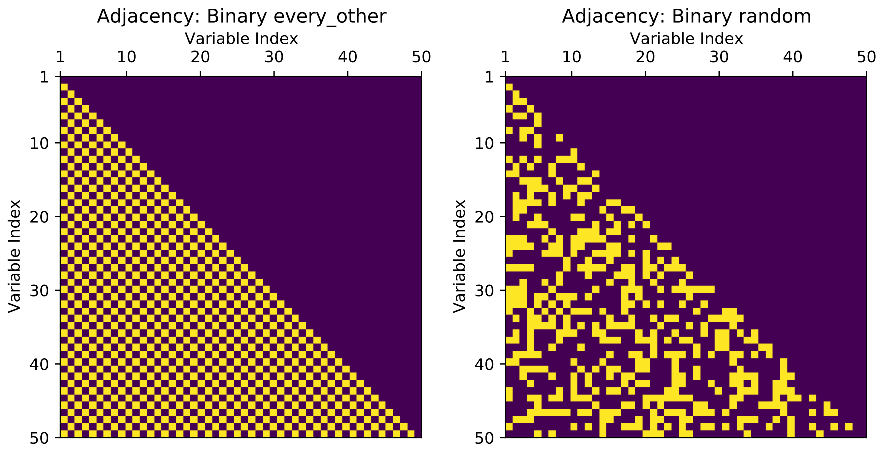


Figure 8. Adjacency matrix used to generate datasets for binary experiments. *Left:* every_other generation scheme. Note in our actual experiments, we used a 800-dimensional version of this adjacency structure. The 50-dimensional matrix is shown here for illustration only. *Right:* random_sparse generation scheme. Conditionally dependent variables are shown in yellow. These adjacency matrices are also used to generate StrNN mask matrices.

We observe that based on the autoregressive assumptions, the synthetic data generating process should draw each x_i as a Bernoulli random variable (i.e., a coin flip) based on $x_1, ..., x_{i-1}$, for $i = 1, ..., d$. Given an adjacency matrix $A \in \{0, 1\}^{d \times d}$, the general structure equations are given by:

$$x_i \sim \text{Bernoulli}(p_i),\ p_i = \text{Sigmoid}(\sum_{j=1}^{i-1} \alpha_{ij} x_j + c_i), \tag{8}$$

where $\alpha_{ij} = 0$ if $A_{ij} = 0$, otherwise $\alpha_{ij} \sim \mathcal{N}(0, 1)$ and $c_i \sim \mathcal{N}(0, 1)$.

Analogous to binary synthetic data generation, for Gaussian data, we sample each variable as:

$$x_i \sim \mathcal{N}(\mu_i, \sigma_i) \text{ where } \mu_i = \sum_{j=1}^{i-1} \alpha_{ij} x_j + c_i,\ \sigma_i \sim \mathcal{N}(0, 1), \tag{9}$$

where $\alpha_{ij} = 0$ if $A_{ij} = 0$, otherwise $\alpha_{ij} \sim \mathcal{N}(0, 1)$.

C.2. Experiment Setup

In this section, we describe the experimental setup for the binary and Gaussian density estimation tasks reported in Section 4.

Method Selection. We compared StrNN using the greedy mask factorization algorithm (Algorithm 1) to the fully autoregressive MADE baseline as proposed in (Germain et al., 2015). MADE serves as a natural baseline since both methods use the outputs of an autoencoder to parameterize marginal probabilities, while our StrNN method has the added capability of enforcing additional conditional independence properties.

Training. Each model is trained with the AdamW optimizer with a batch size of 200 for a maximum of 5000 epochs.

Hyperparameters. We employed a grid search to find the optimal hyperparameters for StrNN and MADE respectively, where the grid is provided in Table 5. The number of hidden layers is varied during the hyperparameter search, and the number of hidden units in each hidden layer is determined by the input dimension d times the hidden size multiplier of that layer. The Best hyperparameters for each model, dataset, and sample size discussed in Sections 4.1 and 4.2 are not listed here since there are too many combinations. Please refer to the code repositories for reproducing the results.

Evaluation Metrics. Results from binary experiments are reported in terms of the negative log-likelihood (NLL) in Figures 2

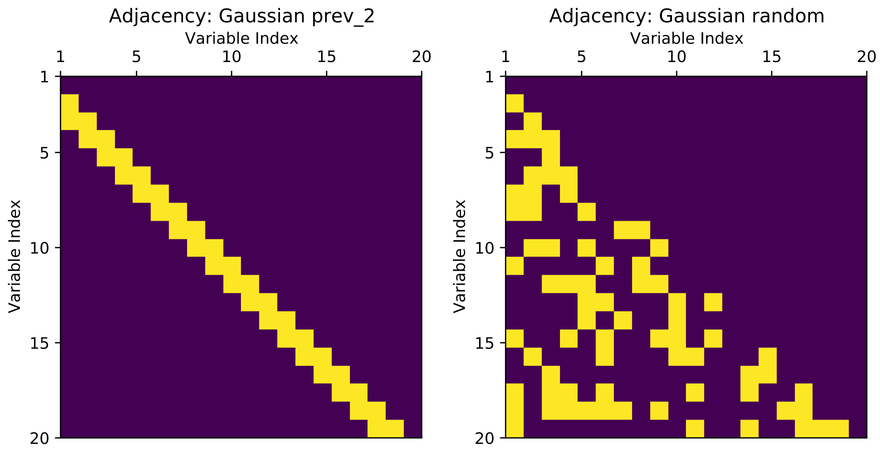


Figure 9. Adjacency matrices used to generate the Gaussian synthetic dataset. Matrices are also used to generate StrNN masks during density estimation tasks. Conditionally dependent variables are shown in yellow. *Left:* prev_2 generation scheme *Right:* random generation scheme.

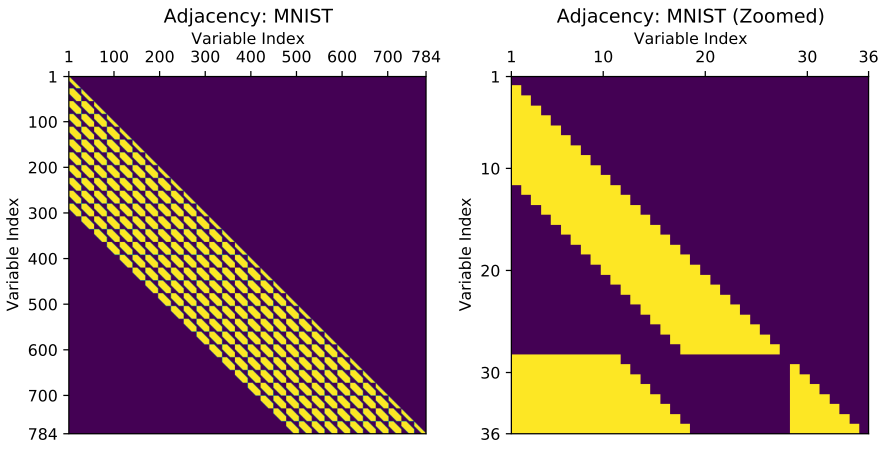


Figure 10. Adjacency matrix used to mask StrNN for the binary MNIST density estimation task, with neighbourhood size set to 10 (`nbr_size=10`). Conditionally dependent variables are shown in yellow. *Left:* All variables. *Right:* Zoomed in view of first 36 variables for illustrative purposes.

and 3. Note that in the binary case, the NLL can simply be rewritten as the binary cross-entropy loss:

$$-\log p(\mathbf{x}) = \sum_{j=1}^{d} -\log p(x_j|\mathbf{x}_{<j}) = \sum_{j=1}^{d} -x_j \log \hat{x}_j - (1-x_j)\log(1-\hat{x}_j). \tag{10}$$

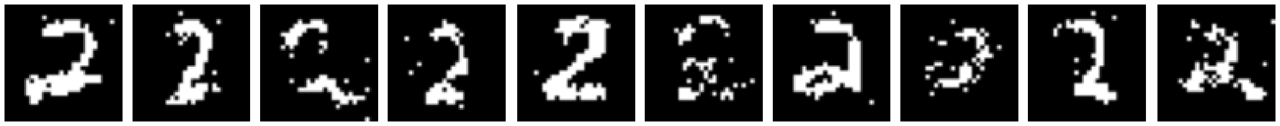
The results from the Gaussian experiments are also reported in NLL, which is calculated by using the neural network outputs as the parameters in each marginal conditional of the Gaussian distribution. The error ranges for the results from these experiments are computed as standard deviation across samples in the held-out test set.

C.3. Additional Results

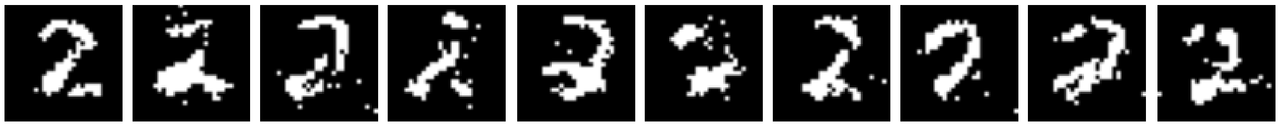
To validate the sample generation quality of StrNN when trained on the MNIST dataset, we display select samples generated by both StrNN and the MADE baseline in Figure 11. We show that even in the low data regime (models fitted on 1000 training samples), both models generate samples with reasonable quality. Hence, we observe that for the density estimation task, injecting prior structure using a StrNN improves likelihood values for each sample under the model without sacrificing generative quality.

Hyperparameter	Grid Values
Activation	[relu]
Epsilon	[1, 0.01, 1e-05]
Hidden size multiplier 1	[1, 4, 8, 12]
Hidden size multiplier 2	[1, 4, 8, 12]
Hidden size multiplier 3	[1, 4, 8, 12]
Hidden size multiplier 4	[1, 4, 8, 12]
Hidden size multiplier 5	[1, 4, 8, 12]
Number of hidden layers	[1, 2, 3, 4, 5]
Learning rate	[0.1, 0.05, 0.01, 0.005, 0.001]
Weight decay	[0.1, 0.05, 0.01, 0.005, 0.001]

Table 5. Hyperparameter grid: StrNN vs. MADE



Top: Samples generated from trained MADE model



Bottom: Samples generated from trained StrNN model with `nbr_size=10`.

Figure 11. Sample MNIST handwritten digits (label 2) generated by MADE and StrNN trained on 1000 data points. Both models generate samples of reasonable quality, while StrNN achieves higher likelihoods at the test samples, as illustrated in Figure 2.