Benchmarking Invertible Architectures on Inverse Problems

Jakob Kruse [1] **Lynton Ardizzone** [1] **Carsten Rother** [1] **Ullrich Köthe** [1]

Abstract

Recent work demonstrated that flow-based invertible neural networks are promising tools for solving ambiguous inverse problems. Following up on this, we investigate how ten invertible architectures and related models fare on two intuitive, low-dimensional benchmark problems, obtaining the best results with coupling layers and simple autoencoders. We hope that our initial efforts inspire other researchers to evaluate their invertible architectures in the same setting and put forth additional benchmarks, so our evaluation may eventually grow into an official community challenge.

1. Introduction

Both in science and in everyday life, we often encounter phenomena that depend on hidden properties $\mathbf{x}$, which we would like to determine from observable quantities $\mathbf{y}$. A common problem is that many different configurations of these properties would result in the *same* observable state, especially when there are far more hidden than observable variables. We will call the mapping f from hidden variables $\mathbf{x}$ to observable variables $\mathbf{y} = f(\mathbf{x})$ the *forward process*. It can usually be modelled accurately by domain experts. The opposite direction, the *inverse process* $\mathbf{y} \to \mathbf{x}$, is much more difficult to deal with. Since $f^{-1}(\mathbf{y})$ does not have a single unambiguous answer, a proper inverse model should instead estimate the full *posterior* probability distribution $p(\mathbf{x}\,|\,\mathbf{y})$ of hidden variables $\mathbf{x}$ given the observation $\mathbf{y}$.

Recent work (Ardizzone et al., 2019) has shown that flow-based invertible neural networks such as RealNVP (Dinh et al., 2016) can be trained with data from the forward process, and then used in inverse mode to sample from $p(\mathbf{x}\,|\,\mathbf{y})$ for any $\mathbf{y}$. This is made possible by introducing additional latent variables $\mathbf{z}$ that encode any information about $\mathbf{x}$ *not* contained in $\mathbf{y}$. Assuming a perfectly representative training set and a fully converged model, they prove that the generated distribution is equal to the true posterior.

[1]Visual Learning Lab, Heidelberg University, Germany. Correspondence to: Jakob Kruse <jakob.kruse@iwr.uni-heidelberg.de>.

First workshop on *Invertible Neural Networks and Normalizing Flows* (ICML 2019), Long Beach, CA, USA

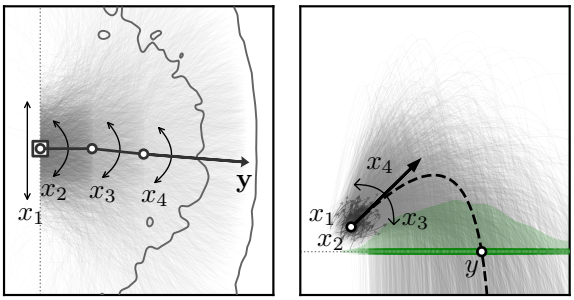


Figure 1. Prior distributions of the parameters $\mathbf{x}$ in either benchmark. *Left:* An articulated arm with three segments is mounted on a rail. x_1 determines the vertical position on the rail and $x_{2\ldots4}$ determine the angles at the three joints. 97% of end points of the resulting arms fall within the contour labelled $\mathbf{y}$. *Right:* An object is thrown upwards and to the right from a starting position (x_1, x_2), at an angle x_3 and initial velocity x_4. We observe the locations of impact y where each trajectory hits the ground, i.e. the x axis. A green curve shows the density of these positions.

Interestingly, this proof carries over to all models offering an exact inverse upon convergence. This poses a natural question: How well can various network types approximate this ideal behavior in practice? Fundamentally, we can distinguish between *hard invertibility*, where the architecture ensures that forward and backward processing are exact inverses of each other (e.g. RealNVP), and *soft invertibility*, where encoder and decoder only become inverses upon convergence (e.g. autoencoders). The former pays for guaranteed invertibility with architectural restrictions that may harm expressive power and training dynamics, whereas the latter is more flexible but only approximately invertible.

We propose two simple inverse problems, one geometric and one physical, for systematic investigation of the resulting trade-offs. Common toy problems for invertible networks are constrained to two dimensions for visualization purposes (Behrmann et al., 2018; Grathwohl et al., 2018). The 4D problems shown here are more challenging, facilitating more meaningful variance in the results of different models. However, they still have an intuitive 2D representation (Fig. 1) and are small enough to allow computation of ground truth posteriors via rejection sampling, which is crucial for proper evaluation. We test ten popular network variants on our two problems to address the following questions: (i) Is soft invertibility sufficient for solving inverse

problems? (ii) Do architectural restrictions needed for hard invertibility harm performance? (iii) Which architectures and losses give the most accurate results?

2. Methods

Invertible Neural Networks (INNs). Our starting point is the model from (Ardizzone et al., 2019), which is based on RealNVP, i.e. affine coupling layers. They propose to use a standard L2 loss for fitting the network's $\mathbf{y}$-predictions to the training data,

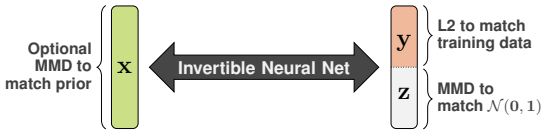
$$\mathrm{L2}(\mathbf{y}) = (\mathbf{y} - \mathbf{y}_{\mathrm{gt}})^2, \tag{1}$$

and an MMD loss (Gretton et al., 2012) for fitting the latent distribution $p(\mathbf{z})$ to $\mathcal{N}(\mathbf{0}, \mathbf{I})$, given samples:

$$\begin{aligned}\mathrm{MMD}(\mathbf{z}) = &\mathbf{E}_{i,j}[\kappa(\mathbf{z}^{(i)}, \mathbf{z}^{(j)})] - 2 \cdot \mathbf{E}_{i,j}[\kappa(\mathbf{z}^{(i)}, \mathbf{z}_{\mathrm{gt}}^{(j)})] + \\ &\mathbf{E}_{i,j}[\kappa(\mathbf{z}_{\mathrm{gt}}^{(i)}, \mathbf{z}_{\mathrm{gt}}^{(j)})]\end{aligned} \tag{2}$$

With weighting factors α, β, their training loss becomes

$$\mathcal{L}(\mathbf{y}, \mathbf{z}) = \mathrm{L2}(\mathbf{y}) + \alpha \cdot \mathrm{MMD}(\mathbf{z}). \tag{3}$$



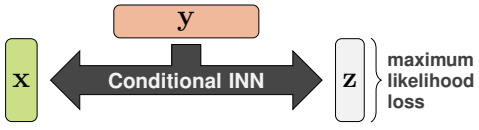
We find that it is also possible to train the network with just a maximum likelihood loss (Dinh et al., 2016) by assuming $\mathbf{y}$ to be normally distributed around the ground truth values $\mathbf{y}_{\mathrm{gt}}$ with very low variance σ^2,

$$\begin{aligned}\mathcal{L}(\mathbf{y}, \mathbf{z}) = &\tfrac{1}{2} \cdot \left(\tfrac{1}{\sigma^2} \cdot (\mathbf{y} - \mathbf{y}_{\mathrm{gt}})^2 + \mathbf{z}^2\right) - \\ &\log \left|\det J_{\mathbf{x} \mapsto [\mathbf{y}, \mathbf{z}]}\right|,\end{aligned} \tag{4}$$

and we compare both approaches in our experiments.

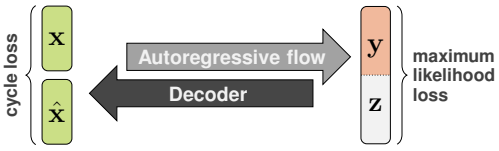
Conditional INNs. Instead of training INNs to predict $\mathbf{y}$ from $\mathbf{x}$ while transforming the lost information into a latent distribution, we can train them to transform $\mathbf{x}$ directly to a latent representation $\mathbf{z}$ *given* the observation $\mathbf{y}$. This is done by providing $\mathbf{y}$ as an additional input to each affine coupling layer, both during the forward and the inverse network passes. cINNs work with larger latent spaces than INNs and are also suited for maximum likelihood training:

$$\mathcal{L}(\mathbf{z}) = \tfrac{1}{2} \cdot \mathbf{z}^2 - \log \left|\det J_{\mathbf{x} \mapsto \mathbf{z}}\right|, \tag{5}$$

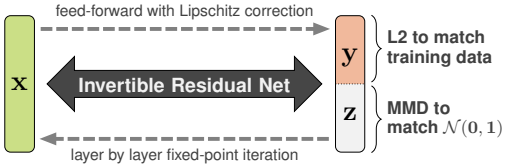


Autoregressive flows. Masked autoregressive flows (MAF) decompose multi-variate distributions into products of 1-dimensional Gaussian conditionals using the chain rule of probability (Papamakarios et al., 2017). Inverse autoregressive flows (IAF) similarly decompose the latent distribution (Kingma et al., 2016). To obtain asymptotically invertible architectures, we add standard feed-forward networks for the opposite direction in the manner of Parallel WaveNets (Oord et al., 2018) and train with Eq. (4) and a cycle loss:

$$\begin{aligned}\mathcal{L}(\mathbf{y}, \mathbf{z}, \hat{\mathbf{x}}) = &\tfrac{1}{2} \cdot \left(\tfrac{1}{\sigma^2} \cdot (\mathbf{y} - \mathbf{y}_{\mathrm{gt}})^2 + \mathbf{z}^2\right) - \\ &\log \left|\det J_{\mathbf{x} \mapsto [\mathbf{y}, \mathbf{z}]}\right| + \alpha \cdot (\mathbf{x} - \hat{\mathbf{x}})^2\end{aligned} \tag{6}$$

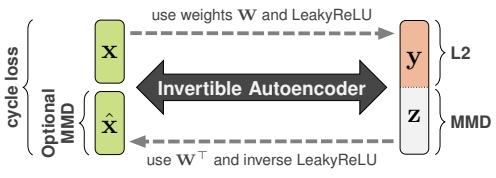


Invertible Residual Networks. A more flexible approach is the i-ResNet (Behrmann et al., 2018), which replaces the heavy architectural constraints imposed by coupling layers and autoregressive models with a mild Lipschitz-constraint on its residual branches. With this constraint, the model's inverse and its Jacobian determinant can be estimated iteratively with a runtime vs. accuracy trade-off. Finding that the estimated Jacobian determinants' gradients are too noisy [1], we train with the loss from Eq. (3) instead.



Invertible Autoencoders. This model proposed by (Teng et al., 2018) uses invertible nonlinearities and orthogonal weight matrices to achieve efficient invertibility. The weight matrices start with random initialization, but converge to orthogonal matrices during training via a cycle loss:

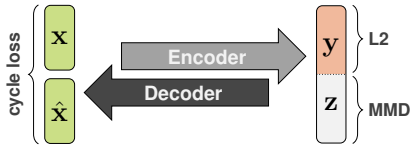
$$\mathcal{L}(\mathbf{y}, \mathbf{z}, \hat{\mathbf{x}}) = \mathrm{L2}(\mathbf{y}) + \alpha \cdot \mathrm{MMD}(\mathbf{z}) + \beta \cdot (\mathbf{x} - \hat{\mathbf{x}})^2 \tag{7}$$



Standard Autoencoders. In the limit of zero reconstruction loss, the decoder of a standard autoencoder becomes the exact inverse of its encoder. While this approach uses

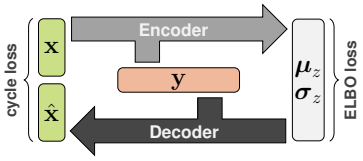
[1] While accurate determinants may be found numerically for toy problems, this would not scale and thus is of limited interest.

two networks instead of one, it is not subject to any architectural constraints. In contrast to standard practice, our autoencoders do not have a bottleneck but use encodings with the same dimension as the input (exactly like INNs). The loss function is the same as Eq. (7).



Conditional Variational Autoencoders. Variational autoencoders (Kingma & Welling, 2013) take a Bayesian approach and thus should be well suited for predicting distributions. Since we are interested in conditional distributions and it simplifies training in this case, we focus on the conditional VAE proposed by (Sohn et al., 2015), with loss

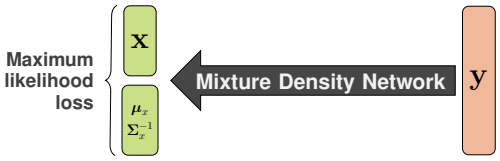
$$\mathcal{L}(\boldsymbol{\mu}_z, \boldsymbol{\sigma}_z, \hat{\mathbf{x}}) = (\mathbf{x}-\hat{\mathbf{x}})^2 - \tfrac{1}{2}\alpha\cdot(1 + \log\boldsymbol{\sigma}_z - \boldsymbol{\mu}_z^2 - \boldsymbol{\sigma}_z). \tag{8}$$



Mixture Density Networks (MDNs). MDNs (Bishop, 1994) are not invertible at all, but model the inverse problem directly. To this end, the network takes $\mathbf{y}$ as an input and predicts the parameters $\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x^{-1}$ of a Gaussian mixture model that characterizes $p(\mathbf{x}\,|\,\mathbf{y})$. It is trained by maximizing the likelihood of the training data under the predicted mixture models, leading to a loss of the form

$$\mathcal{L}(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x^{-1}) = \tfrac{1}{2}\cdot(\mathbf{x}\boldsymbol{\mu}_x^\top\cdot\boldsymbol{\Sigma}_x^{-1}\cdot\mathbf{x}\boldsymbol{\mu}_x) - \log|\boldsymbol{\Sigma}_x^{-1}|^{\frac{1}{2}}. \tag{9}$$

We include it in this work as a non-invertible baseline.



3. Benchmark Problems

We propose two low-dimensional inverse problems as test cases, as they allow quick training, intuitive visualizations and ground truth estimates via rejection sampling.

3.1. Inverse Kinematics

First is the geometrical example used by (Ardizzone et al., 2019), which asks about configurations of a multi-jointed 2D arm that end in a given position, see Fig. 1 left. The forward process takes a starting height x_1 and the three joint angles x_2, x_3, x_4, and returns the coordinate of the arm's end point $\mathbf{y} = [y_1, y_2]$ as

$$y_1 = l_1\sin(x_2) + l_2\sin(x_3-x_2) + l_3\sin(x_4-x_2-x_3)+x_1$$
$$y_2 = l_1\cos(x_2) + l_2\cos(x_3-x_2) + l_3\cos(x_4-x_2-x_3)$$

with segment lengths $l_1 = \frac{1}{2},\ l_2 = \frac{1}{2}$ and $l_3 = 1$.

Parameters $\mathbf{x}$ follow a Gaussian prior $\mathbf{x} \sim \mathcal{N}(\mathbf{0},\ \boldsymbol{\sigma}^2\cdot\mathbf{I})$ with $\boldsymbol{\sigma}^2 = [\frac{1}{16}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}]$. The inverse problem is to find the distribution $p(\mathbf{x}\,|\,\mathbf{y}^*)$ of all arm configurations $\mathbf{x}$ that end at some observed 2D position $\mathbf{y}^*$.

3.2. Inverse Ballistics

A similar, more physically motivated problem in the 2D plane arises when an object is thrown from a starting position (x_1, x_2) with angle x_3 and initial velocity x_4. This setup is illustrated in Fig. 1, right. For given gravity g, object mass m and air resistance k, the object's trajectory $\mathbf{T}(t)$ can be computed as

$$T_1(t) = x_1 - \frac{v_1 m}{k}\cdot\left(e^{-\frac{kt}{m}} - 1\right)$$
$$T_2(t) = x_2 - \frac{m}{k^2}\cdot\left(gm + v_2 k\cdot\left(e^{-\frac{kt}{m}} - 1\right) + gtk\right)$$

with $v_1 = x_4\cdot\cos x_3$ and $v_2 = x_4\cdot\sin x_3$. We define the location of impact as $y = T_1(t^*)$, where t^* is the solution of $T_2(t^*) = 0$, i.e. the trajectory's intersection with the x_1-axis of the coordinate system (if there are two such points we take the rightmost one, and we only consider trajectories that do cross the x_1-axis). Note that here, y is one-dimensional.

We choose the parameters' priors as $x_1 \sim \mathcal{N}(0,\ \frac{1}{4}),\ x_2 \sim \mathcal{N}(\frac{3}{2},\ \frac{1}{4}),\ x_3 \sim \mathcal{U}(9°,\ 72°)$ and $x_4 \sim \text{Poisson}(15)$.

The inverse problem here is to find the distribution $p(\mathbf{x}\,|\,y^*)$ of all throwing parameters $\mathbf{x}$ that share the same observed impact location y^*.

4. Experiments

To compare all approaches in a fair setting, we use the same training data, train for the same number of batches and epochs and choose layer sizes such that all models have roughly the same number of trainable parameters (∼3 M).

We quantify the correctness of the generated posteriors in two ways, using 1000 unseen conditions $\mathbf{y}^*$ obtained via prior and forward process. Firstly, we use MMD (Eq. (2), (Gretton et al., 2012)) to compute the *posterior mismatch* between the distribution $\hat{p}(\mathbf{x}\,|\,\mathbf{y}^*)$ generated by a model and a ground truth estimate $p_{\text{gt}}(\mathbf{x}\,|\,\mathbf{y}^*)$ obtained via rejection sampling:

$$Err_{\text{post}} = \text{MMD}\big(\hat{p}(\mathbf{x}\,|\,\mathbf{y}^*),\ p_{\text{gt}}(\mathbf{x}\,|\,\mathbf{y}^*)\big) \tag{10}$$

Table 1. Quantitative results for inverse kinematics benchmark, see Section 4. The first three columns are averaged over 1000 different observations $\mathbf{y}^*$. DIM($\mathbf{z}$) denotes the dimensionality of the latent space. ML LOSS marks models that were trained with a maximum likelihood loss, while $\mathbf{y}$-SUPERVISION marks models that were trained with an explicit supervised loss on the forward process $\mathbf{x} \to \mathbf{y}$.

METHOD	Err_{post} (10)	Err_{resim} (11)	INFERENCE IN MS	DIM($\mathbf{z}$)	ML LOSS	$\mathbf{y}$-SUPERVISION
INN	0.025	0.015	10	●●	✓	✓
INN (L2 + MMD)	0.017	0.086	9	●●		✓
CINN	**0.015**	**0.008**	11	●●●●	✓	
IAF + DECODER	0.419	0.222	**0**	●●●●	✓	✓
MAF + DECODER	0.074	0.034	**0**	●●●●	✓	✓
IRESNET	0.713	0.311	763	●●		✓
INVAUTO	0.062	0.022	1	●●		✓
AUTOENCODER	0.037	0.016	**0**	●●		✓
CVAE	0.042	0.019	**0**	●●		
MDN	0.007	0.012	601	●●●●	✓	

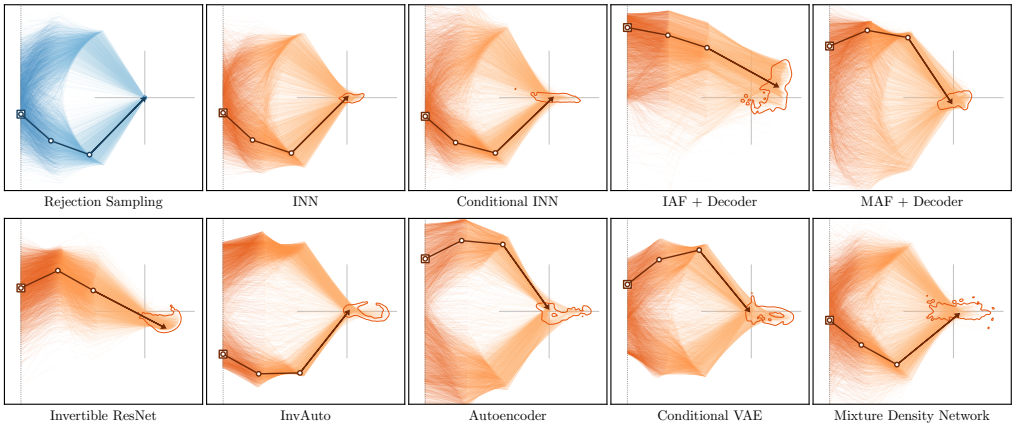


Figure 2. Qualitative results for the inverse kinematics benchmark. The faint lines are arm configurations sampled from each model's predicted posterior $\hat{p}(\mathbf{x} \mid \mathbf{y}^*)$, the target point $\mathbf{y}^* = [1.5, 0]$ is indicated by a gray cross. We emphasize the most likely arm (determined by mean shift) as a bold line. The contour around the target marks the area containing 97% of the sampled arms' end points.

Secondly, we apply the true forward process f to the generated samples $\mathbf{x}$ and measure the *re-simulation error* as the mean squared distance to the target $\mathbf{y}^*$:

$$Err_{\text{resim}} = \mathbb{E}_{\mathbf{x} \sim \hat{p}(\mathbf{x} \mid \mathbf{y}^*)} \| f(\mathbf{x}) - \mathbf{y}^* \|_2^2 \qquad (11)$$

Finally, we report the inference time for each implementation using one *GTX 1080 Ti*.

4.1. Inverse Kinematics

Quantitative results for the kinematics benchmark are shown in Table 1 (extra detail in Fig. 4), while qualitative results for one challenging end point $\mathbf{y}^*$ are plotted in Fig. 2.

Architectures based on coupling layers (INN, cINN) achieve the best scores on average, followed by the simple autoencoder. The invertible ResNet exhibits some mode collapse, as seen in Fig. 2, bottom left. Note that we were unable to train our iResNet-implementation with the estimated Jacobian determinants, which were too inaccurate, and resorted to the loss from Eq. (3). Similarly we would expect the autoregressive models, in particular IAF, to converge much better with more careful tuning.

MDN on the other hand performs very well for both error measures. Note however that a full precision matrix $\mathbf{\Sigma}_x^{-1}$ is needed for this, as a purely diagonal $\mathbf{\Sigma}_x = \mathbf{I}\boldsymbol{\sigma}_x$ fails to model the potentially strong covariance among variables x_i. Since $\mathbf{\Sigma}_x^{-1}$ grows quadratically with the size of $\mathbf{x}$ and a matrix inverse is needed during inference, the method is very slow and does not scale to higher dimensions.

4.2. Inverse Ballistics

Quantitative results for the ballistics benchmark are shown in Table 2 (extra detail in Fig. 5), while qualitative results for one representative impact location y^* are plotted in Fig. 3.

Again we see INN, cINN and the simple autoencoder perform best. Notably, we could not get the conditional VAE to predict proper distributions on this task; instead it collapses to some average trajectory with very high posterior mismatch. The invertible ResNet does better here, perhaps due to the more uni-modal posteriors, but IAF and MAF again fail to capture the distributions properly at all.

Due to the presence of extreme outliers for the error measures in this task, the averages in Table 2 are computed with

Table 2. Quantitative results for the inverse ballistics benchmark. Rows and columns have the same meaning as in Table 1.

METHOD	Err_{post} (10)	Err_{resim} (11)	INFERENCE IN MS	DIM(**z**)	ML LOSS	y-SUPERVISION
INN	**0.047**	**0.019**	21	●●●	✓	✓
INN (L2 + MMD)	0.060	3.668	21	●●●		✓
CINN	**0.047**	0.437	22	●●●●	✓	
IAF + DECODER	0.323	3.457	**0**	●●●●	✓	✓
MAF + DECODER	0.213	1.010	**0**	●●●●	✓	✓
IRESNET	0.084	0.091	307	●●●		✓
INVAUTO	0.156	0.315	1	●●●		✓
AUTOENCODER	0.049	0.052	1	●●●		✓
CVAE	4.359	0.812	**0**	●●●		
MDN	0.048	0.184	175	●●●●	✓	

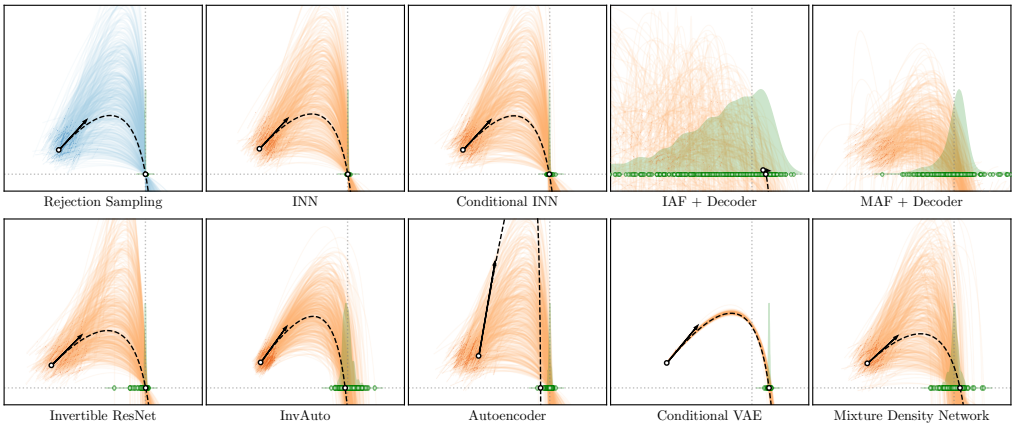


Figure 3. Qualitative results for the inverse ballistics benchmark. Faint lines show the trajectories of sampled throwing parameters and as above, bold is the most likely one. A vertical line marks the target coordinate $y^* = 5$, the distribution of actual impacts is shown in green.

clamped values and thus somewhat distorted. Fig. 5 gives a better impression of the distribution of errors. There the INN trained with Eq. (4) appears the most robust model (smallest maximal errors), followed by the autoencoder. cINN and iResNet come close in performance if outliers are ignored.

5. Discussion and Outlook

In both our benchmarks, models based on RealNVP (Dinh et al., 2016) and the standard autoencoder take the lead, while other invertible architectures seem to struggle in various ways. Success in our experiments was neither tied to maximum likelihood training, nor to the use of a supervised loss on the forward process.

We are aware that training of some models can probably be improved, and welcome input from experts to do so. In the future, the comparison should also include ODE-based methods like (Chen et al., 2018; Grathwohl et al., 2018), variants of Parallel WaveNet (Oord et al., 2018) and classical approaches to Bayesian estimation such as MCMC. Ideally, this paper will encourage the community to join our evaluation efforts and possibly set up an open challenge with additional benchmarks and official leader boards. To this end, we will also publish our code in the coming weeks.

Acknowledgements

Jakob Kruse, Carsten Rother and Ullrich Köthe received financial support from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation program (grant agreement No 647769). Lynton Ardizzone received funding by the Federal Ministry of Education and Research of Germany project 'High Performance Deep Learning Framework' (No 01IH17002).

References

Ardizzone, L., Kruse, J., Rother, C., and Köthe, U. Analyzing inverse problems with invertible neural networks. In *Intl. Conf. on Learning Representations*, 2019.

Behrmann, J., Duvenaud, D., and Jacobsen, J.-H. Invertible residual networks. *arXiv:1811.00995*, 2018.

Bishop, C. M. Mixture density networks. Technical report, Citeseer, 1994.

Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pp. 6571–6583, 2018.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using Real NVP. *arXiv:1605.08803*, 2016.

Grathwohl, W., Chen, R. T., Betterncourt, J., Sutskever, I., and Duvenaud, D. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv:1810.01367*, 2018.

Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Smola, A. A kernel two-sample test. *Journal of Machine Learning Research*, 13(Mar):723–773, 2012.

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

Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. Improved variational inference with inverse autoregressive flow. In *Advances in Neural Information Processing Systems*, pp. 4743–4751, 2016.

Oord, A., Li, Y., Babuschkin, I., Simonyan, K., Vinyals, O., Kavukcuoglu, K., Driessche, G., Lockhart, E., Cobo, L., Stimberg, F., et al. Parallel WaveNet: fast high-fidelity speech synthesis. In *International Conference on Machine Learning*, pp. 3915–3923, 2018.

Papamakarios, G., Murray, I., and Pavlakou, T. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pp. 2335–2344, 2017.

Sohn, K., Lee, H., and Yan, X. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems 28*, pp. 3483–3491, 2015.

Teng, Y., Choromanska, A., and Bojarski, M. Invertible autoencoder for domain adaptation. *arXiv preprint arXiv:1802.06869*, 2018.

Appendix

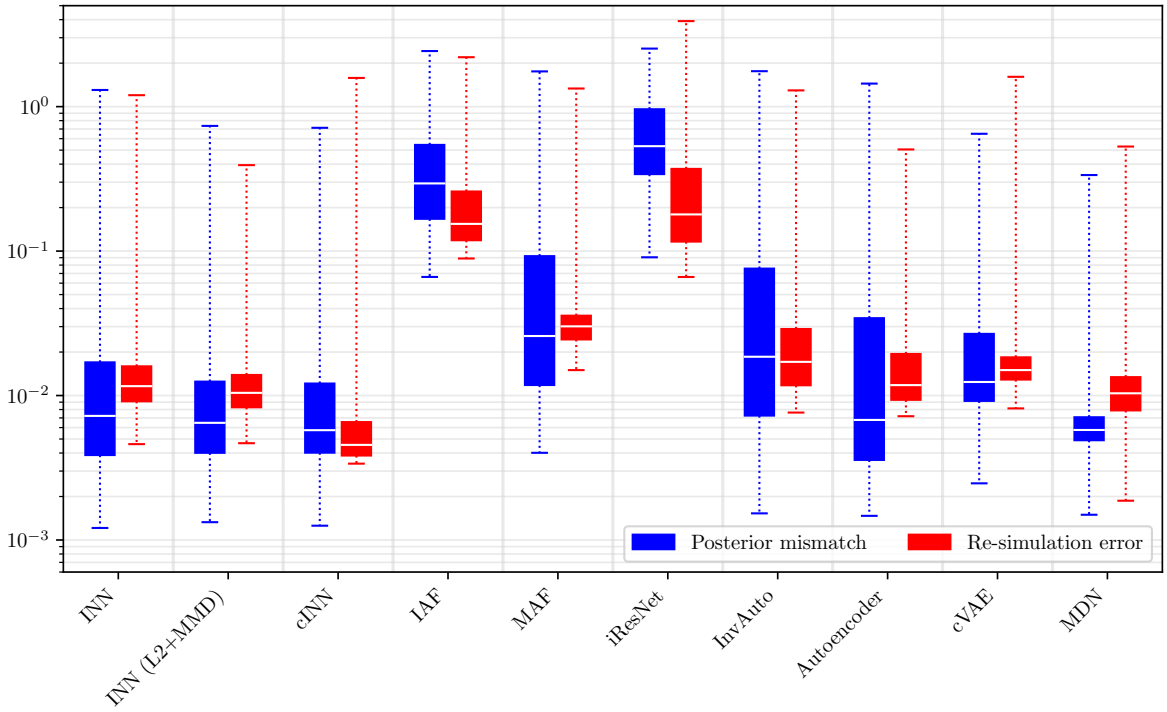


Figure 4. Boxplot of inverse kinematics results from Table 1. The *posterior mismatch* Eq. (10) is shown in blue and the *re-simulation error* Eq. (11) in red. Boxes extend from the lower to upper quartile values of the data and a white line marks the respective median. The dotted lines show the full range of results, including outliers. We use log-scale to accommodate extreme values.

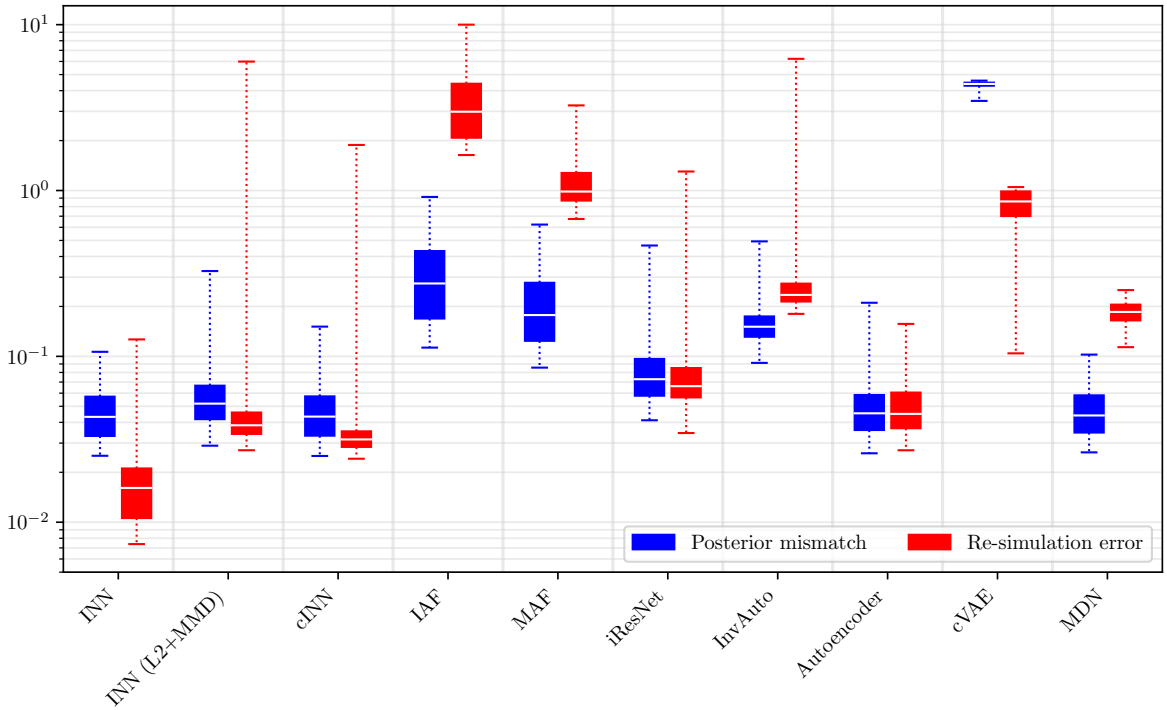


Figure 5. Boxplot of inverse ballistics results from Table 2. The plot follows the same layout as Fig. 5