Autoinverse: Uncertainty Aware Inversion of Neural Networks

Anonymous Author(s)
Affiliation
Address
email

Abstract

Neural networks are powerful surrogates for numerous forward processes. The inversion of such surrogates is extremely valuable in science and engineering. The most important property of a successful neural inverse method is the performance of its solutions when deployed in the real world, i.e., on the native forward process (and not only the learned surrogate). We propose Autoinverse, a highly automated approach for inverting neural network surrogates. Our main insight is to seek inverse solutions in the vicinity of reliable data which have been sampled form the forward process and used for training the surrogate model. Autoinverse finds such solutions by taking into account the predictive uncertainty of the surrogate and minimizing it during the inversion. Apart from high accuracy, Autoinverse enforces the feasibility solutions, comes with embedded regularization, and is initialization free. We verify our proposed method through addressing a set of real-world problems in control, fabrication, and design. Our code and data are available at: https://anonymous.4open.science/r/auto-inverse-BE88/README.md

1 Introduction

"... optimizing for the wrong thing — offering prayers to the bronze snake of data rather the larger force behind it." [6]

With the deep learning breakthrough during the last decade, data-driven modeling using neural network based *surrogates* is now a standard practice in science and engineering. These surrogates strive to imitate the behavior of a *native forward process* (NFP), such as a physics simulation, that maps a *design* into its *performance*[1]. While forward processes are essential for troubleshooting and analysis, oftentimes their ultimate application lies in their inversion, i.e., the reverse process of mapping performances into designs.

Despite the recent progress in inversion of neural networks within multiple disciplines [32, 15, 12, 33], a fundamental unaddressed question is still standing out. Inversion of a surrogate model, fitted to the data sampled from the NFP, is ultimately different than the inversion of the NFP itself. The source of this gap could be explicit, such as the noise in measurements, or implicit, such as the poor sampling of the NFP. Although the obtained solutions from inverting the surrogate can be evaluated on the NFP, none of the current neural inversion methods offers a tailored solution for addressing this important gap.

[1] In computational design, fabrication and robotics, the applications showcased in this paper, *design* refers to the space where the input to the forward process is characterized and parameterized and *performance* refers to the parameterized space of desired properties. Commonly, *hidden state* or *parameters*, and *measurement* or *goal* are used interchangeably with design and performance.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

Our main insight in this work is to *expect* and *account* for any potential mismatch between the data, and consequently the surrogate, on the one hand and the NFP on the other. Our proposed method, `Autoinverse`, realizes this vision by taking into account the predictive uncertainty of the surrogate and minimizing it during the inversion. Therefore, the obtained inversion solutions avoid the unreliable regions within the training data.

We show that our `Autoinverse` strategy can augment existing neural inversion methods (both optimization-based and architecture-based approaches) with uncertainty compensation in a simple and practical manner. `Autoinverse` closes the gap between the surrogate and the NFP *not* through attempting a perfect fit of the surrogate to the NFP, an onerous task, but by finding inverse solutions in the vicinity of the reliable training data where the surrogate and the NFP are most similar. Neural inverse methods equipped with `Autoinverse` outperform their counterparts significantly on both standard data sampled from NFP and imperfect data, e.g., those corrupted by noise. Apart from high accuracy, `Autoinverse` enforces the feasibility solutions, comes with embedded regularization (freeing the inversion approaches from hand-crafted regularizations based on domain knowledge), and is initialization free. It achieves all these properties in a highly automated manner and only with a light, intuitive tuning.

2 Related work

Neural Network Inversion We can divide neural network inversion approaches into two categories: *inverse architectures*, where we compute a network architecture that takes a given performance and maps it into a (distribution of) design(s), and *optimization*, where we optimize for a design such that it produces the desired performance. Although the simplest inverse architecture can be attempted by training a neural network in the reverse direction, it fails because of the one-to-many nature of the mapping. *Tandem* networks use an inverse architecture by employing a pre-trained forward network in order to compute a consistent loss. The tandem approach has been developed independently across different disciplines [21, 29, 30, 35] dating back (at least) to Tominaga [31]. Many inverse architectures try to model the conditional posterior, $p(x|y)$, using variational methods [22, 17] based on (conditional) variational auto-encoders [18]. These networks condition the design on the target performance and yield a distribution of solutions from which multiple samples could be drawn. Kruse et al. [19] show that the invertible neural network (INN), and its conditional variant, give the highest accuracy in terms of both resimulation error and design posterior compared to a wide range of inverse architectures. In this method, a specialized architecture built upon normalizing flows [9] is trained simultaneously in both forward and inverse directions leading to a bijective mapping between design and performance spaces.

When using *direct* [30] optimization, gradient-based optimizers can be readily used as the neural surrogate is differentiable. Ren et al. [28] use stochastic gradient descent via backpropagation with respect to the design variables, and present a highly accurate and practical method. They benchmark their method, dubbed as *neural adjoint* (*NA*), against a set of inverse architectures and obtain significantly more accurate solutions. Sun et al. [30] showed a similar approach except using a quasi-Newton method for optimization. Ansari et al. [2] push forward in this direction by demonstrating that, for piecewise linear neural networks, e.g., those with ReLU activation, the direct optimization can be formulated as a mixed-integer linear program (MILP) and thus obtain *globally* optimal solutions. While optimization methods are very accurate, their main disadvantage is their performance as they can be orders of magnitude slower than inverse architectures.

Neural Networks and Predictive Uncertainty While nerual networks are ubiquitous in almost all branches of natural sciences, their weakness at quantifying predictive uncertainty impedes their use in crucial applications. Using a Bayesian formalism [4], Bayesian neural networks (BNNs) [26, 8, 24], given the training data and a prior over network's parameters, compute the posterior distribution of the parameters. Having computed the posterior, the predictive uncertainty can be computed. The inference step in computing BNNs is known to be computationally hard [16]. This explains the popularity of simpler methods for estimating predictive uncertainty, such as Monte Carlo dropout [10] and Deep Ensembles [20]. Deep Ensembles strikes a good balance between simplicity and practicality on the one hand and predictive performance on the other hand (Section 3.1). Neural networks capable of predictive uncertainty are increasingly adopted in many different applications, such as reinforcement

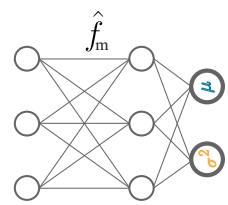
learning and active learning [11]. As we shall see, neural inversion is yet another domain that takes advantage of this trend.

3 Method

`Autoinverse` is an easy-to-implement technique for augmenting neural inverse methods with uncertainty awareness. `Autoinverse` achieves this goal by: 1) Replacing the surrogate with a new surrogate capable of predictive uncertainty [23], and 2) Modifying the inverse cost function in order to not only find accurate designs but also minimize the uncertainty. After reviewing Deep Ensembles as our surrogate (Section 3.1), we introduce two unertainty-aware methods each belonging to the two main categories of inverse methods (Section 3.2 and 3.3).

3.1 Predictive uncertainty using Deep Ensemble

Deep Ensembles comprises of an ensemble of M neural networks $\hat{f}_m$ each capable of a prediction $\mu_m(\mathbf{x})$ and its associated uncertainty $\sigma_m(\mathbf{x})$ (see inset) in form of a Gaussian distribution $\mathcal{N}(\mu_m(\mathbf{x}), \sigma_m(\mathbf{x}))$. The cost function for training each network in the ensemble is the negative log likelihood [27].



$$\mathcal{L}_m^{NLL} := \frac{\log(\sigma_m^2(\mathbf{x}))}{2} + \frac{(\mathbf{y}^* - \mu_m(\mathbf{x}))^2}{2\sigma_m^2(\mathbf{x})}. \tag{1}$$

Intuitively, in case of the *aleatoric uncertainty*, i.e., noisy data, $\mu(\mathbf{x})$ fails to reliably predict $\mathbf{y}^*$. Hence, $\sigma^2(\cdot)$ must increase in order to reduce the loss. The first term ensures $\sigma^2(\cdot)$ does not diverge to infinity.

Next step is to ensemble the results of all the networks into a single prediction and a single uncertainty. Deep Ensembles [20] models the ensemble as a single Gaussian distribution $\mathcal{N}(\mathbb{F}_\mu(\mathbf{x}), \mathbb{F}_\sigma(\mathbf{x}))$ approximating the mixture of M previously computed Gaussian distributions

$$\mathbb{F}_\mu(\mathbf{x}) := \frac{1}{M} \sum_m \mu_m(\mathbf{x}), \tag{2a}$$

$$\mathbb{F}_\sigma(\mathbf{x}) = \frac{1}{M} \sum_m (\sigma_m^2(\mathbf{x}) + \mu_m^2(\mathbf{x})) - \mathbb{F}_\mu^2(\mathbf{x}). \tag{2b}$$

The uncertainty of the ensemble can be decomposed into two uncertainties and be predicted for each input $\mathbf{x}$, aleatoric ($\mathbb{F}_{\sigma A}(\mathbf{x})$) and *epistemic* ($\mathbb{F}_{\sigma E}(\mathbf{x})$). Slight modification to Equation 2b enables us to control the behavior of the neural inversion by tuning the importance of these two uncertainties.

$$\mathbb{F}_\sigma(\mathbf{x}) := \alpha\, \mathbb{F}_{\sigma A}(\mathbf{x}) + \beta\, \mathbb{F}_{\sigma E}(\mathbf{x}), \tag{3a}$$

$$\mathbb{F}_{\sigma A}(\mathbf{x}) := \frac{1}{M} \sum_m \sigma_m^2(\mathbf{x}), \tag{3b}$$

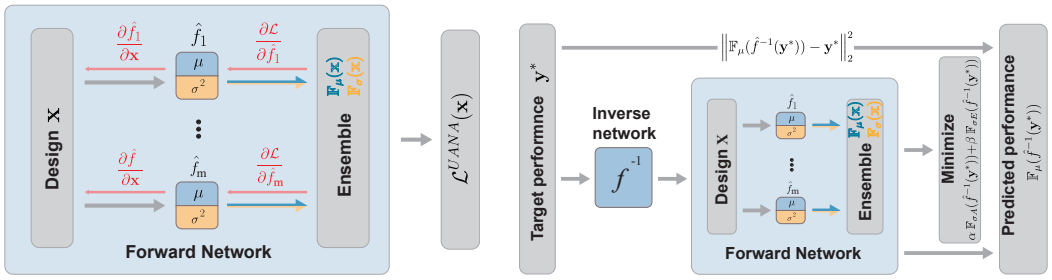
$$\mathbb{F}_{\sigma E}(\mathbf{x}) := \frac{1}{M} \sum_m (\mu_m^2(\mathbf{x}) - \mathbb{F}_\mu^2(\mathbf{x})). \tag{3c}$$

We introduce α and β as hyperparameters to adjust the relative significance of these two uncertainties. Aleatoric uncertainty increases as the noise level in the training data increases. Epistemic uncertainty measures the consensus between the M networks.

3.2 Uncertainty aware neural adjoint (`UANA`)

Neural adjoint (`NA`) [28] uses the cost function $\mathcal{L}^{NA}(\cdot)$ to pushed designs $\mathbf{x}$ to have a performance $\hat{f}(\mathbf{x})$ (using neural surrogate $\hat{f}(\cdot)$) as close as possible to a desired performance $\mathbf{y}^*$:

$$\mathcal{L}^{NA}(\mathbf{x}) := \arg\min_{\mathbf{x}} \left\| \hat{f}(\mathbf{x}) - \mathbf{y}^* \right\|_2^2. \tag{4}$$



(a) (b)

Figure 1: By using a deep ensemble predictive uncertainty estimator as the forward model we can make conventional inversion methods uncertainty aware. On the left we can see `UANA` and on the right `UA-tandem` architecture.

NA uses gradient descent to iteratively reduce the cost function, a scheme much like training neural networks but with the input as the optimization variable instead of network's weights and biases. Equation 5 shows a single `NA` iteration with as the step size.

$$\mathbf{x}^z = \mathbf{x}^{z-1} - \delta(\frac{\partial \mathcal{L}^{NA}}{\partial \hat{f}} \times \frac{\partial \hat{f}}{\partial \mathbf{x}}). \tag{5}$$

We propose to modify `NA` in order to take into account the uncertainty:

$$\mathcal{L}^{UANA}(\mathbf{x}) := \arg\min_{x} \|\mathbb{F}_{\mu}(\mathbf{x}) - \mathbf{y}^*\|_2^2 + \alpha\, \mathbb{F}_{\sigma A}(\mathbf{x}) + \beta\, \mathbb{F}_{\sigma E}(\mathbf{x}) \tag{6}$$

Equation 7 shows how one iteration of `UANA` requires the back-propagation using the ensemble of all gradients of M networks of Deep Ensembles:

$$\mathbf{x}^z = \mathbf{x}^{z-1} - \delta \sum_{m=1}^{M} (\frac{\partial \mathcal{L}^{UANA}}{\partial \hat{f}_m} \times \frac{\partial \hat{f}_m}{\partial \mathbf{x}}) \tag{7}$$

where $\hat{f}_m$ represents one of the networks in the ensemble. Figure 1(a) depicts this collective procedure where each individual network in the ensemble votes for the direction where updating the design will lead to the minimum loss and uncertainty.

3.3 Uncertainty aware tandem (`UA-tandem`)

`Tandem` is a representative of architecture-based methods in which we train an inverse network $\hat{f}^{-1}(\cdot)$ in a manner resembling the encoder-decoder ([29–31]). Unlike the encoder-decoder approach, we start with training the forward model $\hat{f}(\cdot)$. Then we freeze the trainable parameters of $\hat{f}(\cdot)$ and add the inverse model ($\hat{f}^{-1}(\cdot)$) in the position of the encoder and train it to decrease the cost function:

$$\mathcal{L}^{T}(\hat{f}^{-1}(\mathbf{y}^*)) := \arg\min_{\hat{f}^{-1}(\cdot)} \left\| \hat{f}(\hat{f}^{-1}(\mathbf{y}^*)) - \mathbf{y}^* \right\|_2^2. \tag{8}$$

Once $\hat{f}^{-1}(\cdot)$ is trained we can simply query it to find the designs with our desired performance:

$$\hat{f}^{-1}(\mathbf{y}^*) = \mathbf{x}. \tag{9}$$

`Autoinverse` modifies `tandem` by replacing $\hat{f}(\cdot)$ with $\mathbb{F}_{\mu}(\cdot)$, and considering the uncertainty $\mathbb{F}_{\sigma}(\cdot)$:

$$\mathcal{L}^{UAT}(\hat{f}^{-1}(\mathbf{y}^*)) := \arg\min_{\hat{f}^{-1}(\cdot)} \left\| \mathbb{F}_{\mu}(\hat{f}^{-1}(\mathbf{y}^*)) - \mathbf{y}^* \right\|_2^2 + \alpha\, \mathbb{F}_{\sigma A}(\hat{f}^{-1}(\mathbf{y}^*)) + \beta\, \mathbb{F}_{\sigma E}(\hat{f}^{-1}(\mathbf{y}^*)). \tag{10}$$

Figure 1(b) depicts the architecture of `UA-tandem`.

3.4 Implementation

Following [27], we take a 3-step procedure for implementing deep ensemble predictive uncertainty. In practice, training the ensemble networks directly with negative log likelihood loss (Equation 1) is challenging. Instead, we start with training an ensemble of conventional networks with diverse activation functions and MSE norm as its loss. These networks are in fact the initialization of $\mathbb{F}_\mu(\cdot)$. The next step is training $\mathbb{F}_\sigma(\cdot)$ and fine tuning $\mathbb{F}_\mu(\cdot)$ jointly with the negative log likelihood loss (Equation 1). Finally, we replace $\hat{f}(\cdot)$ with $\mathbb{F}_\mu(\cdot)$ and incorporate $\mathbb{F}_\sigma(\cdot)$ in the `Autoinverse` loss.

4 Evaluation

We evaluate the performance of `Autoinverse` via evaluating the existing neural inverse methods and their uncertainty-aware counterparts. In the paper, we focus on `NA` and `UANA` while `tandem` and `UA-tandem` are evaluated mainly in Supplementary Materials (SM). We perform the evaluation on a set of applications in robotics, and fabrication-oriented computational design. We report the error by running the experiments 3 times to capture the variations. More details are provided in each case study and in SM.

Equations 6 and 10 have two hyperparameters (α and β) that keep the balance between the MSE, the aleatoric uncertainty and the epistemic uncertainty. We observe that with a larger epistemic uncertainty we obtain better results. Exploiting this intuition, we tune these parameters for 3 different sets of values for $\{\alpha, \beta\}$: $\{\{0.1, 1\}, \{1, 10\}, \{10, 100\}\}$. We then make two finer step depending on the outcome of the former evaluation and choose the best set of weights. We keep this budget of 5 experiment runs for rival methods as well. Typically, we use 10% of the target performance for tuning our inverse methods.

4.1 Experiments

Multi-joint robot is a simple inverse kinematics problem which is being used as a standard test for neural inverse problems [19, 28]. In this problem, the *design* is the 1D position of the base of the multi-joint robot along with the 1D rotation of its three joints. The inverse problem concerns finding a combination of positions and angles for the base and the joints such that the tip of the arm lands the target position, i.e., the *performance*). We follow [28] for setting up this experiment and its corresponding analytical equation used as the NFP. The training data consists of 10,000 pair of samples generated by randomly sampling the NFP.

Spectral printer Spectral printing enables digital fabrication of the object's appearance faithfully ([13, 5]). Unlike reproduction of the color (e.g., RGB), reproducing the spectrum ensures that the original and the duplicate remain visually similar independent of the color of the light source. Spectral printing has various important applications specially in fine art reproduction using both 2D and 3D printing [25, 1, 29]. Deep neural networks are becoming the main computational tool for modeling the spectral printing process specially when dealing with a large number of inks. The final objective is to find the correct ink densities at each pixel that can best estimate the target 31D spectrum.

In this experiment, we create the NFP by simulating a printer using an ensemble of 20 neural networks. The design space comprises the ink densities and our `spectral printer` NFP predicts the spectrum of the resulting color. We use real, measured data from an 8-channel printer with 8 EPSON inks including Cyan (C), Magenta (M), Yellow (Y), Black (K), Light black (LK), Light light black (LLK), Light Cyan (LC), and Light Magenta (LM). The light inks, added to the standard CMYK to improve the print quality, introduce significant multi-modality. All networks in the ensemble NFP are trained on 40,000 printed patches [1] consisting of different ink-density combinations and their corresponding spectra. The ensembling is intended for an accurate NFP and is independent of our use of Deep Ensembles for computing the uncertainty. The visual nature of `spectral printer` makes it attractive for analyzing different methods. We release this NFP to the public to add another example to the neural inversion testbed.

Soft robots are made of soft, flexible materials. This unique property has made them suitable candidates for interaction with humans in applications like minimally invasive surgery and advanced prosthetics [7]. Unlike `multi-joint robot` with a limited number of rotating joints, each segment

Table 1: The NFP and surrogate errors (mean $\pm$ STD) of different neural inverse methods on `multi-joint robot` for 1000 target locations.

	`NA`	`UANA`	`tandem`	`UA-tandem`	`MINI`	`INN`
NFP error	$(3.24 \pm 0.51) \times 10^{-4}$	$(\mathbf{3.21 \pm 1.48}) \times \mathbf{10^{-6}}$	$(4.42 \pm 1.56) \times 10^{-3}$	$(\mathbf{4.04 \pm 0.38}) \times \mathbf{10^{-5}}$	1.6×10^{-3}	$(9.48 \pm 0.021) \times 10^{-3}$
Surrogate error	$(1.99 \pm 0.05) \times 10^{-8}$	$(9.13 \pm 6.08) \times 10^{-7}$	$(8.58 \pm 3.00) \times 10^{-6}$	$(7.10 \pm 0.64) \times 10^{-6}$	0	$(2.04 \pm 0.017) \times 10^{-13}$

of the soft robot is a potential actuator that through their contraction and expansion can determine the robot's final shape. This inverse kinematics problem is typically solved through partial differential equations ([34]). In order to accelerate the solve time of this inverse kinematics problem, Sun et al. [30] proposed a neural surrogate modeling of the problem and its inversion via `tandem`. The design space in this problem consists of the contraction or expansion of 40 controllable soft edge segments. The superposition of all the actuations determines the final deformation position of the soft robot via the position of its 206 vertices.

We use an FEM-based simulation ([34, 14] as our NFP. We model the relationship between the actuations and the final shape of the soft robot with a neural-network surrogate. Our goal is to solve the neural inversion to find a suitable set of actuations (design) that brings the tip of the soft robot to the target position (performance). The training data consists of 50,000 samples queried by random sampling the actuation with an expansion ratios between -0.2 and 0.2 [30]. The designs are then evaluated on the FEM-based NFP to calculate their deformations.

4.2 Quantitative comparison of different neural inversion methods

We evaluate the accuracy of a set of neural inversion methods on `multi-joint robot` in terms of both the NFP and the surrogate errors. The surrogate error is the difference between the prediction of the neural network and the target performance. The NFP error is the difference between the target and the performance of the generated design evaluated on the NFP. In addition to the core methods described so far, we evaluate mixed-integer neural inversion (`MINI`) [2], and invertible neural networks (`INN`) [3].

Table 1 summarizes the inversion results on 1000 randomly sampled target locations for the multi-joint robotic arm. We keep our evaluation fair by setting the capacity of the neural surrogates comparable wherever possible. For instance, all methods except `MINI` have around 3 million parameters (see SM for more details). We also kept the fairness in tuning stage. Methods with hyper parameters like `UANA`, `UA-tandem` are tuned in 5 stages by hand. Alternatively, the methods without hyper parameters (`NA`, `tandem`) are given $5\times$ extra budget for inversion. We run `NA` and `tandem` $5\times$ and choose the model that generates the best NFP error. We repeat this process $3\times$ and report the standard error. `MILP` and `INN` are fundamentally different in nature. `MILP` finds the global optimum and as a result does not need tuning. `INN` has a latent space which we can sample to generate diverse designs that can generate our target. We sample the latent space 1024 times for all 1000 targets then we evaluate them on the NFP and report the best NFP error. We repeat this process three times to generate the standard error.

Table 1 shows the outstanding accuracy of the inverse methods that adopt `Autoinverse`, i.e., `UANA` and `UA-tandem`, in terms of the NFP error. It also demonstrate how even a perfect surrogate error (e.g., `MINI`) does not guarantee accurate solutions when tested on the NFP. A second look at Table 1 reveals further interesting insights. Although `NA` obtains notably lower surrogate error than its uncertainty-aware counterpart (`UANA`), it performs significantly worse in terms of the NFP error. The main reason for this trend is that, when optimizing the surrogate, `UANA` is not only concerned with finding accurate designs leading to a small accuracy gap between the target and candidate performances, i.e., the surrogate error, but also with those designs featuring low uncertainty (through the uncertainty term in the cost function in Equations 3). Therefore, `UANA` achieves high accuracy in terms of the essential NFP error at the cost of increasing the inconsequential surrogate error. Furthermore, `UA-tandem`, for example, achieves better performance than `NA`. This is a remarkable result for an architecture-based inverse method that is tremendously faster than the optimization-based `NA`.

In terms of computation time, `UANA` takes roughly $3\times$ longer during inversion compared to `NA`. The longer *training* time for the ensemble is a less important concern as it is trivially parallelizable. `UA-tandem` on the other hand takes roughly 10 times longer than `tandem` to train. However, the

Table 2: The distribution of ink densities (≥ 0.4) after the inversion of `spectral printer` using `UANA`. Once we insert noise into LC channel or sample it sparsely, `Autoinverse` detects and avoids it. STD has been rounded to nearest integer.

Model	Data set	NFP error	C	M	Y	K	LC	LM	LK	LLK
UANA	Standard	$(6.30 \pm 0.031) \times 10^{-3}$	186 ± 2	67 ± 4	63 ± 6	3 ± 0	437 ± 2	356 ± 7	26 ± 3	348 ± 5
	Sparse	$(5.64 \pm 0.017) \times 10^{-3}$	316 ± 1	60 ± 2	59 ± 1	2 ± 0	$\mathbf{0 \pm 0}$	326 ± 5	25 ± 4	323 ± 11
	Noisy	$(6.13 \pm 0.026) \times 10^{-3}$	263 ± 1	67 ± 4	34 ± 2	1 ± 0	$\mathbf{0 \pm 0}$	276 ± 3	29 ± 3	378 ± 5
NA	Standard	$(1.57 \pm 0.001) \times 10^{-1}$	1895 ± 13	1060 ± 26	1795 ± 18	162 ± 11	865 ± 8	1396 ± 30	179 ± 15	2378 ± 19
	Sparse	$(1.34 \pm 0.007) \times 10^{-1}$	905 ± 6	606 ± 7	1515 ± 8	137 ± 6	1604 ± 11	2130 ± 17	294 ± 15	2312 ± 24
	Noisy	$(1.47 \pm 0.002) \times 10^{-1}$	1192 ± 14	988 ± 10	1128 ± 10	55 ± 6	1029 ± 23	948 ± 21	283 ± 12	1742 ± 37

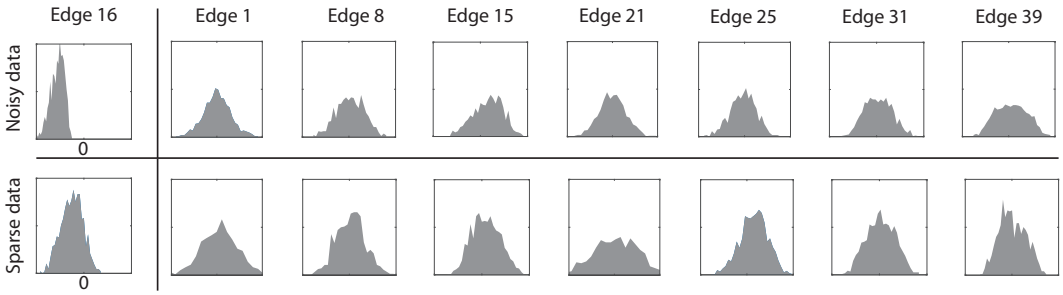


Figure 2: Distribution of actuation values for `soft robot`. Edge 16 is corrupted by noise (top) and sampled sparsely (bottom) in positive range. While other randomly chosen edges feature both negative and positive actuations, `UANA` produces solutions without (or with less) positive actuations for Edge 16.

training time is amortized during the inversion of many samples using the inverse architecture. We report extensively the time performance and the details of the training in SM.

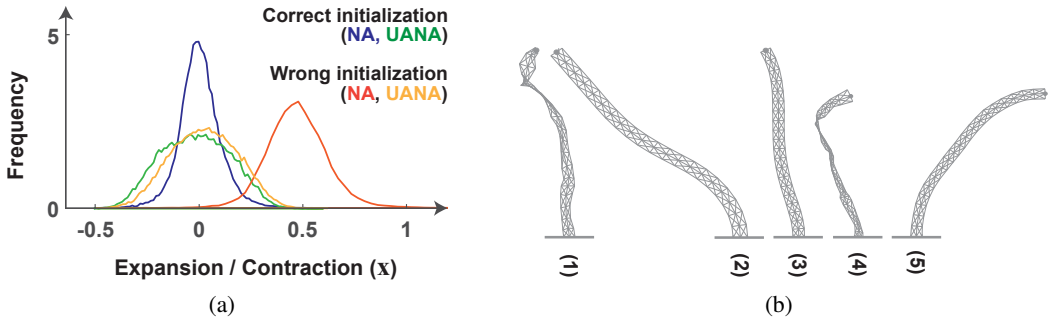
4.3 Neural inversion in the presence of imperfect data

One of the main advantages of `Autoinverse` appears in scenarios when the training data suffers from noise, e.g., due to noise in measurement or poor sampling, e.g., due to sparsity in some regions. We evaluate the performance of `Autoinverse` on imperfect training data using `NA` and `UANA` (`tandem` and `UA-tandem` are evaluated under the same configuration in the SM).

Locally sparse data We sample both `soft robot` and `spectral printer` NFPs (Section 4.1) in a way that the data does not contain any samples from one of the inputs in a specific interval. For `spectral printer`, we would like to find ink densities to reproduce the spectra of the colors in the painting in Figure 4 (made of 3568 color spectra). We sample the printer channels at 0 (no ink), 0.05, 0.1, 0.5, and 1 (full ink) densities to form the *standard* training data (with no known uncertainty). We create a partially *sparse* dataset similar to the standard one except for the Light Cyan (LC) channel for which we only have samples at 0, 0.05 and 0.1. Table 2 shows that while for the standard dataset `UANA` finds inverse solutions that include the LC channel frequently (437 times), for the sparse dataset it avoids this channel *completely* and compensate for it using the Cyan channel. `UANA` is able to avoid this channel as the epistemic uncertainty increase in sparse regions of dataset (see SM for more details).

In `soft robot` we sample its 16th (among 40) controllable edge only in negative range (contraction only). We then use the trained network to invert 1000 test samples. Figure 2 shows the distribution of each edge for 1000 inversion tasks. We have plotted the distribution for the 16th edge as well as for 7 other randomly chosen edges. As evident from Figure 2 bottom row, the inversion was reluctant to choose designs with positive actuation for this edge.

Locally noisy data set With the same problem configuration as before, we would like to test the robustness of `Autoinverse` on a data set locally corrupted with noise. We start with a standard data set and inject Gaussian noise $\mathcal{N}(0,\ 0.1)$ to the spectrum of the samples with more than 0.4 LC density. Table 2 shows how after introducing noise to light cyan, the network avoids that channel and compensates it by using more cyan instead. In `soft robot` we corrupt the final shape of the soft



(a) (b)

Figure 3: On the left we can see the distribution of actuations calculated by `UANA` and `NA` with two different initialization: one initialization near the training data distribution and one far from it. On the right we can see a range of randomly chosen soft robot shapes calculated by different methods with different regularization and initialization.

robot with Gaussian noise $\mathcal{N}(0,\ 0.5)$ for all the shapes where the 16th edge has positive actuation. As we can see in Figure 2 (top row), the inversion has completely avoided positive actuation for this edge.

4.4 Autoinverse brings AutoML to neural inversion

Autoinverse incorporates feasibility. Deep Ensembles produces high epistemic uncertainty outside the distribution of the training data. This includes the regions where the NFP is not defined and thus not sampled. For example, ink densities outside $[0, 1]$ are not printable. If such cases arise during inversion, they are clipped to $[0, 1]$ [1]. In such regions, networks in the ensemble do not agree and thus the epistemic uncertainty increases. `Autoinverse` automatically avoids these unfeasible regions. Revisiting the painting reproduction experiment using `spectral printer`, we observe how `NA`, after optimizing for the ink densities, their clipping and feeding them to `spectral printer` NFP, results in a poor reproduction of the original painting (Figure 4 left). `UANA`, on the other hand, achieves spectacular reproduction accuracy. This is explained by the plot in Figure 4 right, showing the distribution of ink densities obtained from both methods.

Autoinverse has built-in regularization. We incorporate regularization into inversion methods in order to obtain solutions that, among other things, agree with the observations and follow a certain statistical distribution. Oftentimes, regularization is case specific, requires human knowledge, and comes with unexpected side effects. Here we show that `Autoinverse` follows the distribution of the training data naturally, by taking into account the epistemic uncertainty. We validate this point using both `spectral printer` and `soft robot` experiments. We show that `Autoinverse` without any explicit regularization performs better or on par with its counterpart inversion methods that incorporate regularization.

In the `spectral printer` experiment, we evaluate the effect of the *boundary loss*, originally proposed as a generic regularization for `NA` to limit the designs in a box constraint (see [28] and SM). The boundary loss is added to Equation 4 and weighted using a hyperparameter. We tune this parameter with the same tuning budget we allocate for tuning the uncertainty weights (5 set of inversions on evaluation data). In Figure 4, we observe that although `NA` with boundary loss improves the distribution of ink densities within the valid region ($[0, 1]$), it still trails the regularization-free `UANA` significantly.

Regularization is less intuitive for `soft robot` as the superposition of all actuations determines the final shape and whether it is physically plausible. In [30], the regularization is a smoothness term that keeps actuation values near each other (see SM). Figure 3(b)(1) demonstrates how `NA` fails without regularization to control the robot with a reasonable deformation. Once the regularization is added to `NA`, the designs become physically meaningful (Figure 3(b)(2)). Figure 3(b)(3) shows how `UANA` performs comparably without any regularization.

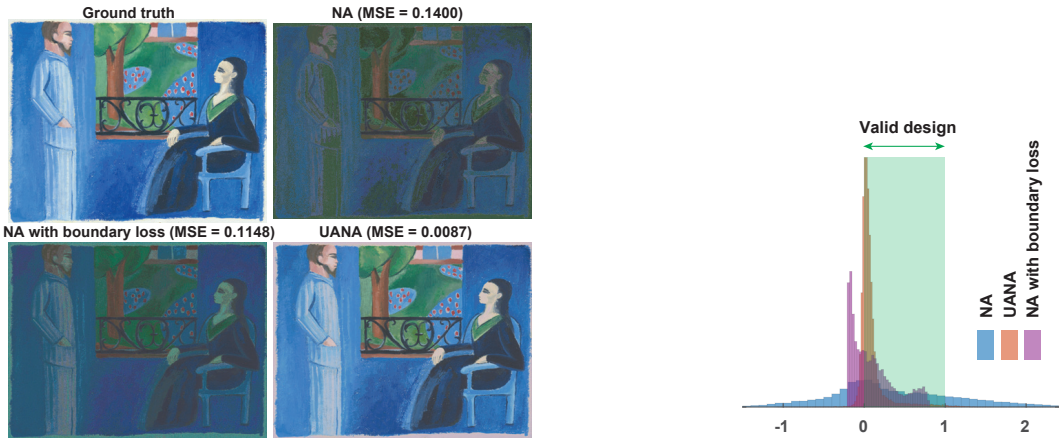


Figure 4: In painting reproduction, UANA outperforms NA with and without a boundary loss term. The distribution of the inverse solutions is shown on right.

Autoinverse is initialization free. Sensitivity to the initialization is a widely known issue in non-convex optimizations. Despite equipping NA with the smoothness term in `soft robot`, Figure 3(a) demonstrates how an incorrect initialization can result in solutions with seemingly good surrogate and regularization loss (see SM) but in the infeasible region of design space (Figure 3(b)(4)). UANA with the same incorrect initialization *and* without any regularization, leads to soft robot designs that reach the target location accurately and produce plausible deformations (Figure 3(b)(5)). For UANA, the solver first reduces the main contributor to the optimization objective, i.e., the epistemic uncertainty. Once a region with a reasonably small uncertainty is reached, the accuracy term starts to take effect and a desirable solution within the valid range of the design space is found.

4.5 Ablation studies

Separating the ensembling effect Although in both NA and UANA we use surrogates with similar capacity (number of network parameters), one could argue that the higher performance of the UANA come from the ensemble architecture of its surrogate. We detach the impact of ensembling and uncertainty awareness on the inversion performance by implementing `NA ensemble` where, instead of a single, high-capacity forward neural network, it uses an ensemble of networks. The inversion procedure is kept identical to NA (see SM). We employ `NA ensemble` in `multi-joint robot` with the same configurations as in Section 4.2. The surrogate and NFP error for `NA ensemble` are $(3.30 \pm 0.59) \times 10^{-9}$ and $(1.17 \pm 0.32) \times 10^{-4}$, respectively. Comparing these values with those in Table 1, `NA ensemble` shows a slight improvement over NA but is significantly outperformed by UANA.

Diversity of activation functions in Deep Ensembles Originally Deep Ensembles uses a seimialr set of networks with identical activation function [20]. In practice, we observe that a diverse set of activation layers leads to a better performance of `Autoinverse`. Different activation functions generate different behaviours, show higher disagreement where the training data is under-represented and, thus, result in a clear landscape of epistemic uncertainty. For all experiments in this work, we use a diverse range of activation functions (see SM). As an ablation, we run UANA on `spectral printer` using sparse data (same configuration as Section 4.3) but with ReLU as the only activation layer. In contrast to Table 2, UANA (with ReLU only activation) does not completely avoid the sparse domain and delivers 19 ± 4 solutions that contain LC with densities ≥ 0.4.

5 Discussion

We look forward to see `Autoinverse` adopted for more inverse architectures, beyond `tandem` and more optimizations (beyond first-order NA). The `Autoinverse` cost function is multi-objective. It is highly interesting to capture the trade-off between accuracy and uncertainty using the Pareto front.

References

[1] Navid Ansari, Omid Alizadeh-Mousavi, Hans-Peter Seidel, and Vahid Babaei. Mixed integer ink selection for spectral reproduction. *ACM Transactions on Graphics (TOG)*, 39(6):1–16, 2020.

[2] Navid Ansari, Hans-Peter Seidel, and Vahid Babaei. Mixed integer neural inverse design. *arXiv preprint arXiv:2109.12888*, 2021.

[3] Lynton Ardizzone, Jakob Kruse, Carsten Rother, and Ullrich Köthe. Analyzing inverse problems with invertible neural networks. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJed6j0cKX`.

[4] José M Bernardo and Adrian FM Smith. *Bayesian theory*, volume 405. John Wiley & Sons, 2009.

[5] Yongda Chen, Roy S Berns, and Lawrence A Taplin. Six color printer characterization using an optimized cellular yule-nielsen spectral neugebauer model. *Journal of Imaging Science and Technology*, 48(6):519–528, 2004.

[6] Brian Christian and Tom Griffiths. *Algorithms to live by: The computer science of human decisions*. Macmillan, 2016.

[7] Matteo Cianchetti, Cecilia Laschi, Arianna Menciassi, and Paolo Dario. Biomedical applications of soft robotics. *Nature Reviews Materials*, 3(6):143–153, 2018.

[8] John S Denker and Yann LeCun. Transforming neural-net output levels to probability distributions. In *Proceedings of the 3rd International Conference on Neural Information Processing Systems*, pages 853–859, 1990.

[9] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[10] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

[11] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep bayesian active learning with image data. In *International Conference on Machine Learning*, pages 1183–1192. PMLR, 2017.

[12] Konstantinos Gavriil, Ruslan Guseinov, Jesús Pérez, Davide Pellis, Paul Henderson, Florian Rist, Helmut Pottmann, and Bernd Bickel. Computational design of cold bent glass façades. *ACM Transactions on Graphics (TOG)*, 39(6):1–16, 2020.

[13] Roger David Hersch and Frédérique Crété. Improving the yule-nielsen modified neugebauer model by dot surface coverages depending on the ink superposition conditions. In *Color Imaging X: Processing, Hardcopy, and Applications*, volume 5667, pages 434–447. SPIE, 2005.

[14] Thomas JR Hughes. *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation, 2012.

[15] Jiaqi Jiang, Mingkun Chen, and Jonathan A Fan. Deep neural networks for the evaluation and design of photonic devices. *Nature Reviews Materials*, 6(8):679–700, 2021.

[16] Laurent Valentin Jospin, Wray Buntine, Farid Boussaid, Hamid Laga, and Mohammed Bennamoun. Hands-on bayesian neural networks–a tutorial for deep learning users. *arXiv preprint arXiv:2007.06823*, 2020.

[17] Yashar Kiarashinejad, Sajjad Abdollahramezani, and Ali Adibi. Deep learning approach based on dimensionality reduction for designing electromagnetic nanostructures. *npj Computational Materials*, 6(1):1–12, 2020.

[18] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[19] Jakob Kruse, Lynton Ardizzone, Carsten Rother, and Ullrich Köthe. Benchmarking invertible architectures on inverse problems. *arXiv preprint arXiv:2101.10763*, 2021.

[20] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *arXiv preprint arXiv:1612.01474*, 2016.

[21] Dianjing Liu, Yixuan Tan, Erfan Khoram, and Zongfu Yu. Training deep neural networks for the inverse design of nanophotonic structures. *ACS Photonics*, 5(4):1365–1369, 2018.

[22] Wei Ma, Feng Cheng, Yihao Xu, Qinlong Wen, and Yongmin Liu. Probabilistic representation and inverse design of metamaterials based on a deep generative model with semi-supervised learning strategy. *Advanced Materials*, 31(35):1901111, 2019.

[23] David JC MacKay. Probable networks and plausible predictions-a review of practical bayesian methods for supervised neural networks. *Network: computation in neural systems*, 6(3):469, 1995.

[24] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors. *Machine Learning: An Artificial Intelligence Approach, Vol. I.* Tioga, Palo Alto, CA, 1983.

[25] Peter Morovič, Ján Morovič, Jordi Arnabat, and Juan Manuel García-Reyero. Revisiting spectral printing: A data driven approach. In *Color and Imaging Conference*, volume 2012, pages 335–340. Society for Imaging Science and Technology, 2012.

[26] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[27] David A Nix and Andreas S Weigend. Estimating the mean and variance of the target probability distribution. In *Proceedings of 1994 ieee international conference on neural networks (ICNN'94)*, volume 1, pages 55–60. IEEE, 1994.

[28] Simiao Ren, Willie Padilla, and Jordan Malof. Benchmarking deep inverse models over time, and the neural-adjoint method. *arXiv preprint arXiv:2009.12919*, 2020.

[29] Liang Shi, Vahid Babaei, Changil Kim, Michael Foshey, Yuanming Hu, Pitchaya Sitthi-Amorn, Szymon Rusinkiewicz, and Wojciech Matusik. Deep multispectral painting reproduction via multi-layer, custom-ink printing. *ACM Trans. Graph.*, 37(6):271:1–271:15, December 2018.

[30] Xingyuan Sun, Tianju Xue, Szymon Rusinkiewicz, and Ryan P Adams. Amortized synthesis of constrained configurations using a differentiable surrogate. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=wdIDt--oLmV`.

[31] Shoji Tominaga. Color control using neural networks and its application. In *Color Imaging: Device-Independent Color, Color Hard Copy, and Graphic Arts*, volume 2658, pages 253–260. International Society for Optics and Photonics, 1996.

[32] Hsiao-Yu Fish Tung, Adam W Harley, William Seto, and Katerina Fragkiadaki. Adversarial inverse graphics networks: Learning 2d-to-3d lifting and image-to-image translation from unpaired supervision. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 4364–4372. IEEE, 2017.

[33] Jiajun Wu, Ilker Yildirim, Joseph J Lim, Bill Freeman, and Josh Tenenbaum. Galileo: Perceiving physical object properties by integrating a physics engine with deep learning. *Advances in neural information processing systems*, 28, 2015.

[34] Tianju Xue, Alex Beatson, Sigrid Adriaenssens, and Ryan Adams. Amortized finite element analysis for fast pde-constrained optimization. In *International Conference on Machine Learning*, pages 10638–10647. PMLR, 2020.

[35] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]