Functions Inverse Using Neural Networks via Branch-Wise Decomposition and Newton Refinement

Abdullah Balamash
Electrical and Computer Engineering-Faculty of Engineering, King Abdulaziz University, Jeddah, Saudi Arabia

***Abstract*—In this work, a unified framework (using Neural Networks) is proposed to find the inverse of mathematical functions, spanning both simple one-to-one mapping and complex multivalued relations. The approach uses standard multilayer Neural Networks (NN) to approximate the functions' inverse and introduces a deterministic branch-wise decomposition to handle multi-valued inverses. For single-valued (one-to-one) functions, a NN is directly trained on input-output pairs to learn the inverse mapping. For multi-valued functions, the function domain is decomposed into one-to-one branches, and a dedicated NN is trained for each branch. A refinement step using Newton's method is applied to the NN output to further improve inversion accuracy. Across a broad set of benchmark functions, the proposed approach achieved low mean absolute error (MAE) and mean squared error (MSE) in recovering the true inverse, with high round-trip consistency. Newton refinement further reduces inversion error by rapidly converging to higher precision solutions. Notably, even for multi-valued inverse functions, each branch-specific NN can accurately recover the true inverse. Accordingly, standard NN, when combined with branch-wise decomposition and Newton refinement, can serve as an effective universal approximator for the inverse of functions across a spectrum of complexities.**

***Keywords*—Neural networks; function inverse; Newton method; branch-wise decomposition**

I. Introduction

A. Background and Related Works

Finding the inverse function is a fundamental problem in many domains, from control systems and robotics to scientific computing and optimization. Traditional analytical or numerical methods can be challenging to apply when the function is complex or not single-valued. Neural networks offer a powerful alternative, since when they are provided with sufficient training data, they can approximate any well-behaved and invertible continuous function to an arbitrary accuracy [1]. In practice, inversion using NN has been explored in control systems and other fields [2], but significant challenges remain when the inverse mapping is multi-valued [3]. Recent advances in physics-informed neural networks (PINNs) have demonstrated remarkable capability in solving both forward and inverse problems by embedding physical laws directly into the learning process [4], although these approaches primarily address parameter identification rather than multi-valued function inversion.

If the forward function is $y = f(x)$ is not one-to-one, then the inverse relation $x = f^{-1}(y)$ is multi-valued, meaning a single y corresponds to multiple valid solutions (x). For example, both $y = x^2$ and $y = \sin(x)$ have two or more inverse values for a given y. Standard neural regression fails in this setting because training data is not a single-valued mapping: the same input y appears with different target x, leading to contradictory samples [5]. A conventional NN trained on such data will tend to predict the average of the contradictory samples [6]. Previous studies have shown that multi-valuedness can hinder learning [3]. It has been shown in robotics that a standard NN cannot learn a multi-valued kinematics function, necessitating specialized solutions [7]. Recent work using deep learning for inverse kinematics in robotic manipulators has demonstrated both the potential and limitations of various neural architectures in handling such non-unique mappings [8].

Several schemes were explored to cope with multi-valued inverses. Mixture Density Networks (MDNs) model the inverse distribution as a mixture of Gaussian distributions, where each Gaussian component corresponds to a single outcome. MDN minimizes the log-likelihood of the observed data under the predicted mixture model, which allows the model to fit all possible outcomes [9]. While MDNs are powerful, they are complex, hard to train, and less interpretable in practice. They are difficult to train since they need to learn several things at once, such as means, variances, and mixing weights for each Gaussian component. They also need a careful initialization and regularization to avoid both overfitting and underfitting. Recent applications of MDNs in photonic inverse design have shown their effectiveness in handling non-uniqueness, through the challenges of specifying the number of mixture components and joint parameter optimization persist [10]. Transfer learning approaches have been proposed to mitigate some of these training difficulties [11].

The modular neural networks (mixtures of experts) method was explored, where each expert network learns one branch of the inverse mapping, and a gating network decides which expert network to activate for a given input [12], [13]. While such gating architectures can handle multi-valued mappings, they add an extra layer of complexity, as the gating layer must be trained and tuned, and performance can be sensitive to the partitioning of the input space.

Multi-headed neural networks provide another approach, in which a single network outputs several possible inverse candidates (heads) simultaneously [14], [15]. While this approach allows multiple solutions, it still couples all outputs into one model, requiring complex loss functions and additional mechanisms to ensure that each head specializes in a distinct branch. For example, a multiple-choice learning strategy can be employed, wherein for each training example, only the output head closest to the ground truth is updated, forcing other heads

to diverge [16]. This added training complexity is necessary to prevent the heads from collapsing to the output. The work in [15] demonstrated a multi-headed tandem neural network approach for non-uniqueness in inverse design of layered photonic structures, utilizing a self-attention mechanism to constrain and separate multiple outputs.

Beyond these mainstream methods, researchers have explored other techniques for multi-valued inversion. One idea is to provide the network with additional inputs or context to disambiguate the inverse. In study [3], the authors proposed a state-regulated NN that augments a multilayer perceptron with a discrete state variable to indicate different branches of the inverse. This effectively transforms a one-to-many mapping to multiple one-to-one mappings, provided the correct state is supplied during the inference. Some researchers have approached the one-to-many mapping using manifolds or piecewise mappings. In study [17], the researchers present a method that learns locally linear maps on different regions of a data manifold to represent multiple outputs for a given input.

In photonic design, generative models have been leveraged to address the inverse problem of non-unique solutions. For example, conditional generative adversarial networks have been used to generate multiple candidate structures that all produce the same target optical spectrum [18], and conditional variational autoencoders have been applied to learn the distribution of possible inputs that yield a given output [19]. More recently, diffusion models have emerged as a powerful alternative for inverse design problems, demonstrating superior performance in capturing complex distributions and naturally handling one-to-many mappings [20]. In [21], the authors proposed DiffMat, a diffusion model-based framework for energy-absorbing mathematical design that effectively realizes one-to-many mapping from properties to geometries.

Another strategy is to use invertible neural networks (normalizing flows). Invertible networks can be trained simultaneously on forward and inverse mappings, and once trained, can produce a complete distribution of x that correspond to a given y [22]. However, invertible networks and other generative approaches introduce significant complexity and often require substantial training data. They also yield probabilistic rather than deterministic selection of the correct inverse branch in a specific network. In [23], the authors provide a comprehensive review of NN-based regularization methods for inverse problems, highlighting both the theoretical foundations and practical challenges of these approaches.

In contrast, this work adopts a branch-wise deterministic learning approach, where the multi-valued inverse relation is decomposed into distinct single-valued branches, each modeled by a simple neural network. This avoids the probabilistic interpretation and training difficulties of the MDNs, as well as the architectural complexity of mixtures-of-experts and multi-headed models.

B. Deterministic Inverse-Learning Framework

In this work, a simple deterministic framework for learning inverse functions of single-valued and multi-valued cases is proposed through branch-wise decomposition rather than relying on probabilistic modeling.

Unlike probabilistic inverse-learning approaches such as MDNs, invertible neural networks, or generative models, the proposed framework adopts a fully deterministic inverse-learning paradigm. The novelty of this work lies not in introducing a new neural architecture, but in reframing inverse learning as a branch-constrained deterministic problem, where each inverse branch is learned independently and refined using a numerical solver. This formulation ensures branch consistency, avoids output ambiguity, and eliminates the training instability and interoperability limitations commonly associated with probabilistic or multi-head inverse models. Furthermore, the integration of a neural approximator with Newton refinement establishes a hybrid learning-numerical pipeline that combines fast global approximation with high-precision convergence at low inference cost, a capability not explicitly addressed in existing inverse-learning frameworks.

The main contributions of this work are summarized as follows:

- A deterministic inverse-learning framework that reformulates multi-value inverse problems as branch-constrained learning tasks, enabling guaranteed branch consistency and eliminating ambiguity inherent in probabilistic inverse models.
- A hybrid neural-numerical inversion pipeline, where neural networks provide fast global inverse approximations and Newton refinement ensures high-precision convergence with minimal additional computation cost.
- A reusable benchmark and evaluation protocol based on direct inverse error and round-trip consistency, exposing both the strengths and failure modes of purely learning-based or purely numerical inversion strategies.

To clarify the positioning of the proposed framework within the broader landscape of inverse-learning approaches, Table I provides a qualitative comparison with representative methods commonly used for learning inverse mapping. The comparison focuses on key design dimensions that are particularly relevant to practical inverse problems, including the determinism of the output, architectural complexity, training stability, inference cost, and consistency of inverse-branch selection.

The remainder of the paper is organized as follows. In Section II (Methodology), the NN training procedure for inverse functions is formalized, and the branch-wise decomposition and Newton refinement technique are detailed. In Section III (Benchmark Design), the selected test functions and the experimental setup are described. Section IV (Results) presents quantitative outcomes that directly support our claims, demonstrating the successful inversion of the chosen functions, confirming that the branch-wise mechanism effectively solves the multi-valued problem, and showing that Newton's method improves accuracy. A dedicated discussion section (Section V) highlights practical implications and limitations. The paper is concluded in Section VI.

TABLE I. CONCEPTUAL COMPARISON OF REPRESENTATIVE INVERSE-LEARNING APPROACHES

Method	Output Type	Determinism	Training Stability	Inference Cost	Branch-Consistency
MDNs	Probabilistic	No	Low	High	No
Mixture of Experts	Semi-deterministic	No	Medium	Medium	No
Mult-head NNs	Deterministic	No	Low	Medium	No
Invertible NNs	Probabilistic	No	Low	High	No
Proposed Method	Deterministic	Yes	High	Low	Yes

II. METHODOLOGY

This framework considers a forward function $y = f(x)$ where x and $y \in \mathcal{R}$. Our goal is to learn an approximate inverse function ($f^{-1}(y) \approx x$) using a neural network. The focus is restricted to single-output functions, for which the inverse is either single-valued or multi-valued. Each function f in our framework is assumed to be continuous and differentiable, and leverage f and its derivative for data generation and Newton's refinement.

A. Neural Network Architecture

The standard feed-forward multilayer perceptron (MLP) is used as the function approximator to compute the inverse. For each function f, one or more MPLs are configured to take y as input and output an x prediction. A fixed architecture with two hidden nodes, each with 128 neurons, is used. One can use different architectures for different functions to improve performance, but we decided not to do so because our goal is to prove the concept. The tanh activation function is used. For one-to-one functions, a single network is trained on the entire dataset. For many-to-one functions, a separate network is trained for each branch. The loss function is the mean square error between the network's output $\hat{x}$, and the true x. The Adam optimizer is used for training the network.

For many-to-one functions, the function is decomposed into one-to-one branches, and a separate network is trained for each branch. During the prediction, a mechanism is needed to select the appropriate branch. In this work, it is assumed that simple rules can identify branches: for instance, in a real application context, the physical meaning of x indicates which branch to use (e.g., positive vs. negative solution). Alternatively, one could run all branch networks and select the output that satisfies $f(\hat{x}) = y$.

For each benchmark function, a large training dataset is generated by sampling x values from a reasonable range of its domain (covering a range of interest). Then $y = f(x)$ is computed for each sample. These (x, y) pairs are used to train the network with y as input and x as target output. Each network's performance is validated using a test set of (x, y) pairs not seen during training.

It is important to emphasize that the proposed approach does not rely on probabilistic inference, gating networks, or multi-output architectures. Instead, each inverse branch is learned independently using a standard feedforward neural network, resulting in a modular, interpretable, and deployment-friendly design. The deterministic structure simplifies training, avoids mode collapse, and enables predictable inference behavior, which is particularly important in critical and real-time applications.

B. Newton Refinement Procedure

Once the neural network inverse is obtained, one can optionally use the Newton method to refine the solution. The output of the network is considered as the initial guess of the solution (the initial value x_0 of the Newton method. When the network has already achieved high accuracy, Newton updates make minimal adjustments, but when extreme precision is needed, a few iterations can dramatically reduce the error. A limit is set on the number of iterations (one to two iterations). This hybrid approach effectively combines the speed and the generality of neural network learning with the precision of a numerical solver. It is noted that the Newton method requires a good initial guess to converge to the correct root.

Fig. 1 illustrates the workflow of the proposed inverse-learning framework. Given a value y, a deterministic branch-selection step identifies the appropriate inverse branch using prior knowledge, simple rule-based constraints, or a forward-consistency check. The selected branch-specific neural network then provides an initial inverse estimate ($\widehat{x_0}$), which can be further refined using a small number of Newton iterations to improve numerical accuracy. The sequential structure separates branch identification from inverse approximation, ensuring deterministic behavior, interpretability, and efficient inference.

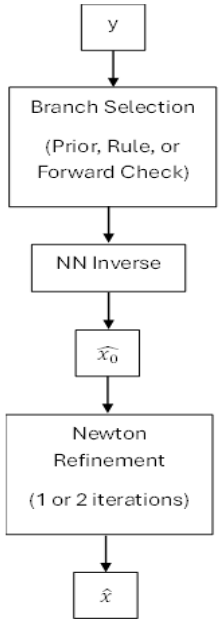

Fig. 1. Overview of the deterministic inverse-learning framework.

III. Benchmark Design

The proposed method is evaluated in a suite of representative functions. Each function is chosen to test specific challenges in inverse learning, such as steep nonlinearities, multiple inverse values, or the absence of a closed-form inverse. The following subsections present the benchmark functions, their characteristics, and the inverse challenges they pose.

A. Benchmark Functions

1) Cubic function ($y = x^3$): This is a strictly increasing and monotonic one-to-one function defined in $\mathbb{R}$. Its inverse is a single-valued, continuous function. Although an analytic inverse exists, this function is used to verify that the MLP can learn a nonlinear invertible function with high accuracy.

2) Quadratic function ($y = x^2$): This function is a two-value inverse defined in $\mathbb{R}$: $x = \pm\sqrt{y}$. The domain of the inverse is restricted to $y \geq 0$. Two branches are created: Branch A for $x \geq 0$ ($x = +\sqrt{y}$) and Branch B for $x \leq 0$ ($x = -\sqrt{y}$). Training data are generated for each branch by sampling x in $[0, X_{max}]$ for Branch A, and $[-X_{max}, 0]$ for Branch B, and then $y = x^2$ is computed. This tests the branch-wise approach on a simple discontinuous inverse.

3) Quartic function ($x = x^4$): This is a two-value inverse defined in $\mathbb{R}$ like the quadratic function. It is defined in $\mathbb{R}$, and the inverse domain is restricted to $y \geq 0$. It has two branches: Branch A for $x \geq 0$ ($x = +\sqrt[4]{y}$) and Branch B for $x \leq 0$ ($x = -\sqrt[4]{y}$). It grows faster than the quadratic functions, and it is flatter near 0, which might impose a tougher learning problem in this region. This function tests the network's ability to learn a steep nonlinear inverse.

4) Exponential function ($y = e^x$): This is a classic one-to-one monotonic and increasing function that grows rapidly, defined in $\mathbb{R}$. The inverse domain is restricted to $y > 0$. Because the function spans many orders of magnitude, Attention is paid to how the network handles small and large y values.

5) Sigmoid function ($y = \frac{1}{1+e^{-x}}$): This is a strictly increasing function from 0 and 1, which expresses a function with a tight range relative to its domain, which is $\mathbb{R}$. It also has a steep slope in the middle. The domain is restricted to $[-L, L]$ since y saturates when x goes to $\pm\infty$ (extremely flat curves).

6) Sine function – principal branch ($y = sin(x)$ for $x \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$): Over the interval $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$, $\sin(x)$ is monotonic and covers $[-1, 1]$ one-to-one. This is considered as the principal branch for $sin^{-1}(y)$. This tests learning an inverse for a smooth but non-linear bounded function.

7) Composite function $y = x + e^x$): This function is strictly increasing for all x in $\mathbb{R}$. It is invertible (its closed-form inverse is known as the Lambert W function). However, its inverse does not have a simple elementary function. This function is included to represent a one-to-one function where the inverse is not a simple one (there is no standard closed inverse form that one can use for evaluation). The difficulty in learning is that the function grows exponentially for positive values of x, and almost linear when x is negative.

B. Data Generation and Performance Metrics

For each function, 50,000 random samples are drawn of x for training, 10000 samples of x for validating the model, and 10000 samples of x for testing. The sampling distribution is chosen to cover the domain of interest uniformly in general, while slightly biasing towards challenging regions (e.g., near boundaries, peak or trough values, steep regions, or flat regions). The test sample is chosen from the same domain of interest, ensuring that no test sample appears in the training sample. The y values are computed using the function of interest ($y = f(x)$). Data normalization is done by scaling y and x to zero mean and unit variance.

The mean-absolute error (MAE) and mean-squared error (MSE) between the predicted inverse $\hat{x}$ and true x are reported. Additionally, the report round-trip error ($|\hat{y} - y|$), where $\hat{y} = f(\hat{x})$ is reported. Both the direct and round-trip errors provide valuable insight. The round-trip error becomes more significant for steep curves, where even a slight deviation in x can amplify it. In contrast, the direct error is more relevant for flat curves, where a large deviation in x has a minor effect on the round-trip error. The suffixes _x and _y are used to distinguish between the direct and the round-trip errors, respectively. For each function, the Newton refinement step is done for very few (one to two) iterations, and the errors are reported before and after applying the Newton method.

IV. Results

The benchmark results for all functions are shown in Table II. The following subsections discuss these results by dedicating a subsection to each function.

TABLE II. Benchmark Performance

benchmark	MAE_x	MSE_x	MAE_y	MAE_y + Newton
Cubic	0.022169	0.001525	0.083884	6.21×10^{12}
Quadratic	0.016777	0.000555	0.071686	0.000519
Quartic	0.041733	0.005616	0.126786	0.051399
Exp	0.028440	0.001271	0.040086	0.000371
Sigmoid	0.094882	0.024346	0.000568	0.000024
Sin	0.019173	0.000867	0.007929	0.000025
x + Exp	0.004318	0.000023	0.015278	0.000025

1) Cubic function ($y = x^3, x \in [-3, 3]$): This function is globally monotonic and therefore invertible. The neural network achieved good accuracy ($MAE_x = 0.022$ and $MAE_y = 0.084$), and a good round-trip error ($MAE\_y = 0.084$). However, Newton refinement caused catastrophic divergence (error > 1e12) because $f'(x) = 3x^2$ vanishes at $x = 0$. This illustrates that while cubic inverses can be learned accurately, Newton steps must be applied carefully with damping or safeguards to avoid division by near-zero derivatives. This result shows that the scheme correctly handles the inverse of simple monotonic functions.

2) Quadratic function ($y = x^2$ for $x \in [0, 5]$): The results of the square function (principal branch) show that the scheme

handles the inversion of this function with high accuracy ($MAE_x = 0.017$ and $MAE_y = 0.072$). After Newton refinement, the round-trip error dropped sharply ($MAE\_y = 0.00052$). The main challenge lies near $x = 0$, where the derivative approaches 0. Nonetheless, the combination of neural approximator and Newton refinement provided highly accurate inverses.

3) Quartic function ($y = x^4$ for $x \in [0, 3]$): The benchmark outcomes of the quartic function (principal branch) demonstrate that the inversion scheme can manage this more challenging polynomial effectively ($MAE\_x = 0.042$). Although, the $MAE\_y = 0.127$. It is slightly high; one step of the Newton refinement reduced it to 0.051. This result shows that the scheme can handle functions with flat and steep curves.

4) Exponential function ($y = e^x$ for $x \in [-3, 3]$): The neural network achieved good accuracy ($MAE\_x = 0.028$ and $MAE\_y = 0.04$). After a single Newton step, the round-trip error dropped by two orders of magnitude (to 0.0004) demonstrating near-perfect recovery.

5) Sigmoid function ($y = \frac{1}{1+e^{-x}}$ for $x \in [-8, 8]$): Although the saturation near 0 and 1 makes the learning a bit difficult, the results are extremely good ($MAE_x = 0.094$) and the $MAE_y = 0.0006$) was extremely small. This is expected since the sigmoid curve is mostly a flat curve. Newton refinement nearly eliminated round-trip error discrepancies ($MAE\_y = 0.0004$).

6) Sine function – principal branch ($y = sin(x)$ for $x \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$): Baseline performance was good ($MAE\_x = 0.019$ and $MAE\_y = 0.0079$). Newton refinement significantly reduced the MAE_y to 0.00024. This demonstrates that oscillatory functions can be inverted accurately when restricted to monotonic branches.

7) Composite function $y = x + e^x$, for $x \in [-4, 3]$): The combined linear and exponential function achieved the best results overall. The errors were extremely low ($MAE_x = 0.0043$ and $MAE_y = 0.015$). After the Newton refinement, the MAE_y dropped to 0.0003.

In summary, the MLP model produced low prediction errors across all functions, with MAE_x generally below 0.1. The subsequent round-trip analysis showed that applying Newton's method to refine the predicted results usually reduced the output errors by one or more orders of magnitude. In 7 out of the eight functions, Newton's method converged to a highly accurate solution, yielding a near-zero error. The cubic function was the sole exception, where Newton's method diverged dramatically, which is an expected outcome when the starting point is not close to a real root. Overall, these results demonstrate that the scheme predictions, combined with Newton refinement, can effectively invert a variety of smooth functions with high accuracy.

V. Discussion

The experimental results demonstrate that the proposed framework achieves reliable inverse approximation across a wide range of nonlinear functions, including monotonic, composite, and multi-valued cases. The branch-wise decomposition effectively resolves the ambiguity inherent in multi-valued inverses, while the Newton refinement step significantly improves precision with minimal computational overhead.

A key observation is that purely numerical solvers such as Newton's method are highly sensitive to initialization, whereas purely learning-based approaches may suffer from limited precision. The hybrid neural-numerical formulation bridges this gap by combining global approximation capability with local convergence guarantees. The cubic function example further highlights the importance of derivative behavior and motivates the use of safeguarded or damped Newton variants in future extensions.

Beyond synthetic benchmarks, the deterministic and modular nature of the proposed framework makes it well-suited for real-world inverse problems in control systems, robotics, and engineering design, where interpretability, predictability, and low inference cost are often more critical than probabilistic diversity.

The proposed framework is currently demonstrated on one-dimensional inverse functions, where branch decomposition can be explicitly defined. While this setting allows precise analysis and benchmarking, extending the approach to higher-dimensional inverse problems introduces challenges related to branch identification, scalability, and Jacobian conditioning. Additionally, the Newton refinement step relies on derivative information and may require safeguarding in regions where derivatives approach zero. These limitations motivate future research on automated branch discovery, damped refinement strategies, and multi-dimensional inverse learning.

VI. Conclusion and Future Work

This work presents a unified framework for learning inverse functions using neural networks, demonstrating that even a simple multilayer perceptron (MLP) can accurately approximate inverse mappings for a single-valued function. For multi-valued cases, the study introduces a branch-wise decomposition method by splitting the inverse relationship into single-valued branches, each modeled by a separate network. This avoids the ambiguity of one-to-many mappings and ensures deterministic outputs. Experiments across a range of benchmark functions showed that each branch network effectively learns its part of the inverse, and the combined branch networks reconstruct the full inverse relationship.

To enhance the precision, the approach integrates Newton's method as a post-processing step, refining the neural network's estimates to achieve near-exact inversion. This hybrid approach merges the speed and the flexibility of neural approximations with the rigor of analytical refinement, making it useful for a wide range of engineering applications.

In conclusion, this work introduces a deterministic and interpretable framework for inverse function learning that combines neural approximation with numerical refinement. By explicitly decomposing multi-values inverses into single-valued branches and integrating Newton's method as a refinement step, the proposed approach achieves accurate, stable, and computationally efficient inversion. The framework addresses practical limitations of existing probabilistic and multi-head

inverse-learning methods, particularly in terms of determinism, deployment simplicity, and precision. These properties make the approach especially relevant for critical, real-time, and resource-constrained applications, while also providing a foundation for future extensions to higher-dimensional inverse problems.

REFERENCES

[1] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," Neural Networks, vol. 2, no. 5, pp. 359–366, 1989.

[2] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," IEEE Transactions on Neural Networks, vol. 1, no. 1, pp. 4–27, 1990.

[3] J.-M. Wu, C.-C. Wu, J.-C. Chen, and Y.-L. Lin, "Set-valued functional neural mapping and inverse system approximation," Neurocomputing, vol. 173, pp. 1276–1287, 2016.

[4] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," Journal of Computational Physics, vol. 378, pp. 686–707, 2019.

[5] Y. Tomikawa, H. Akiyama, and K. Nakayama, "Layered neural network with a feedback to realize a many-valued function," IEICE Technical Report (Neural Computing), NC95-167, 1996.

[6] D. J. MacKay, "Bayesian interpolation," Neural Computation, vol. 4, no. 3, pp. 415–447, 1992.

[7] E. Oyama, A. Agah, K. F. MacDorman, T. Maeda, and S. Tachi, "A modular neural network architecture for inverse kinematics model learning," Neurocomputing, vol. 38–40, pp. 797–805, 2001.

[8] D. Cagigas-Muñiz, "Artificial Neural Networks for inverse kinematics problem in articulated robots," Engineering Applications of Artificial Intelligence, vol. 126, p. 107175, 2023.

[9] C. M. Bishop, "Mixture density networks," Aston University, 1994.

[10] R. Unni, K. Yao, and Y. Zheng, "Deep convolutional mixture density network for inverse design of layered photonic structures," ACS Photonics, vol. 7, no. 10, pp. 2703–2712, 2020.

[11] L. Cheng, P. Singh, and F. Ferranti, "Transfer learning-assisted inverse modeling in nanophotonics based on mixture density networks," IEEE Access, vol. 12, pp. 55218–55224, 2024.

[12] M. I. Jordan and R. A. Jacobs, "Hierarchical mixtures of experts and the EM algorithm," Neural Computation, vol. 6, pp. 181–214, 1994.

[13] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," Neural computation, vol. 3, no. 1, pp. 79–87, 1991.

[14] C. Rupprecht et al., "Learning in an uncertain world: representing ambiguity through multiple hypotheses," in Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2017, pp. 3611–3620.

[15] X. Yuan, S. Wang, L. Gu, S. Xie, Q. Ma, and J. Guo, "Multi-headed tandem neural network approach for non-uniqueness in inverse design of layered photonic structures," Optics & Laser Technology, vol. 176, p. 110972, 2024.

[16] A. Guzmán-Rivera, D. Batra, and P. Kohli, "Multiple choice learning: learning to produce multiple structured outputs," in Advances in Neural Information Processing Systems (NIPS) 25, 2012, pp. 1808–1816.

[17] D.-K. Oh, S.-H. Oh, and S.-Y. Lee, "Learning one-to-many mapping with locally linear maps based on manifold structure," IEEE Signal Processing Letters, vol. 18, no. 9, pp. 521–524, 2011.

[18] P. Dai and others, "Inverse design of structural color: finding multiple solutions via conditional generative adversarial networks," Nanophotonics, vol. 11, no. 13, pp. 3057–3069, 2022.

[19] T. Rahman and others, "Leveraging generative neural networks for accurate, diverse, and robust nanoparticle design," Nanoscale Advances, vol. 7, no. 2, pp. 634–642, 2025.

[20] Y. Mao et al., "Generative adversarial networks and mixture density networks-based inverse modeling for microstructural materials design," Integrating Materials and Manufacturing Innovation, vol. 11, no. 4, pp. 637–647, 2022.

[21] H. Wang, Z. Du, F. Feng, Z. Kang, S. Tang, and X. Guo, "DiffMat: Data-driven inverse design of energy-absorbing metamaterials using diffusion model," Computer Methods in Applied Mechanics and Engineering, vol. 432, p. 117421, 2024.

[22] L. Ardizzone et al., "Analyzing inverse problems with invertible neural networks," arXiv preprint arXiv:1808.04730, 2018.

[23] A. Habring and M. Holler, "Neural-network-based regularization methods for inverse problems in imaging," GAMM-Mitteilungen, vol. 47, no. 4, p. e202470004, 2024.