Autoencoder Enhanced Surrogate Model

José Filipe Afonso 11/04/2025

Introduction

This work presents a progressive Conditional Denoising Autoencoder (CDAE) model designed for efficiently solving steady-state problems associated with plasma-surface interactions. Specifically, the model tackles equilibrium states described mathematically by the deterministic equation:

$$F(\vec{x}, \vec{y}) = 0 \tag{1}$$

Here, $\vec{x}$ represents known input parameters, and $\vec{y}$ denotes corresponding equilibrium outputs.

Problem Formulation and Previous Works

The primary objective is developing a robust surrogate model that accurately predicts equilibrium outputs based on specific input conditions. The equilibrium dataset is generated from solutions of deterministic steady-state differential equations modeling plasma-surface interactions.

So far, the approaches have been explored included predicted $\vec{y}$ directly from the $\vec{x}$ conditions using MLP models, which effectively means that each of the predicted components $\hat{y}_i$ only effectively depends on the connections established between the $\vec{x}$ components and their high-level representations, so that our problem is given by obtain the $f(\vec{x};\theta)$, such that the pairs $(\vec{x}, f(\vec{x}))$ approximately satisfy $F(\vec{x}, f(\vec{x};\theta)) = 0$.

Recently, the projection method was presented in *Physics-consistent machine learning: Output projection onto physical manifolds* . It comprises training a system $(\vec{x}, f(\vec{x};\theta))$ using a MLP model , defining the physical constraints as equations $g(\vec{x}, \vec{y}) = 0$ and then projecting the predicted output $\hat{y}$ onto the constraint manifold by solving the following optimization problem:

$$min_p \parallel p - f(x;\Theta) \parallel_W^2 \quad \text{s.t} \;\; g(x,p) = 0 \tag{2}$$

In this way, the model predictions always comply with the specified physical laws, even if the model itself didn't learn them well during training.

This work follows a different direction. Inspired by the fact that the pairs $(\vec{x}, \vec{y})$ are the fixed point solutions of $F(\vec{x}, \vec{y}) = 0$, we aspire to find a formulation that generally internalizes and learn the structure within $(\vec{x}, \vec{y})$ and not solely stick to connections established between $\vec{x}$ and/or through the constraints $g(\vec{x}, \vec{y}) = 0$.

Dataset

The dataset (*data_3000_points.txt*) is externally sourced from previously published work and made publicly available on GitHub [https://github.com/matildevalente/physics_consistent_machine_learning/tree/main/data/ltp_system]. It comprises 3000 input-output pairs $(\vec{x}, \vec{y})$, capturing equilibrium conditions across a variety of input conditions. In this system, $\vec{x}$ corresponds to the vector of experimental conditions, and $\vec{y}$ corresponds to the vector of the chemical species concentrations. The details of how it was generated and about the physical simulator are widely explained in REF Projection.

Model Architecture: Conditional Denoising Autoencoder (CDAE)

The CDAE model is central to our methodology. It is designed to reconstruct equilibrium states from artificially corrupted versions of the outputs $\vec{y}$, conditioned explicitly on inputs $\vec{x}$. Formally, the CDAE operates as follows:

- **Noisy Inputs**: Given the original equilibrium output y, a corrupted version $\tilde{y}$ is generated by adding noise ε:

$$\tilde{y} = y + \varepsilon, \quad \varepsilon \sim N(0, \sigma^2) \tag{3}$$

- **Conditional Reconstruction**: The CDAE aims to reconstruct y from the noisy observation $\tilde{y}$ conditioned on inputs $\vec{x}$:

$$\hat{y} = g_\phi(\tilde{y}, x) \tag{4}$$

- **Loss Function**: Training involves minimizing the mean squared reconstruction error between the true equilibrium outputs and their CDAE reconstructions:

$$\mathcal{L}_{CDAE}(\phi) = \mathbb{E}[\|g_\phi(\tilde{y}, x) - y\|_2^2] \tag{5}$$

- **Corrective Vector Field**: Post-training, the CDAE implicitly defines a deterministic corrective vector field:

$$v(y; x; \sigma) = g_\phi(y; x; \sigma) - y \tag{6}$$

This vector field guides iterative refinements toward equilibrium solutions.

Moreover, rather than using a fixed noise level, a noise schedule allows the model to see examples with varying levels of corruption during training. This enables the model to learn robust corrections across a spectrum of deviations from the manifold.

Relationship to Denoising Score Matching

Our CDAE approach parallels the principles behind denoising score matching, commonly applied in stochastic generative models. Score matching involves learning gradients of log probability densities from noisy observations. However, instead of learning stochastic gradients, the CDAE directly learns a deterministic correction field, in our deterministic context, Hence, while inspired by denoising score matching, our approach diverges by leveraging deterministic equilibrium conditions rather than stochastic processes.

Moreover, architectures like diffusion and score-based models apply a gradual denoising process (from high to low noise) to reconstruct clean samples. Likewise, a noise schedule helps the CDAE "navigate " and correct trajectories from high-noise (off-manifold) states to low-noise equilibrium states.

Refinement Dynamics

The refinement dynamics process combines an initial prediction and iterative updates driven by a corrective vector field. It includes the following steps:

- **Initial Estimate with Direct Prediction**

$$\hat{y} = f_\theta(x) \tag{7}$$

The function $f_\theta(x)$ provides an initial guess of the equilibrium state given the input x. Although $\hat{y}$ might be close to the manifold, further refinement is needed for convergence.

- **Iterative Refinement**

The model iteratively refines the initial estimate using a learned corrective vector field derived from the denoising autoencoder (CDAE). At iteration k, the state is updated as:

$$y^{(k+1)} = y^{(k)} + \eta\, v(y^{(k)}; x), \quad y^{(0)} = \hat{y} \tag{8}$$

Where $v(y^{(k)}; x) = g_\phi(y^{(k)}; x) - y^{(k)}$ and η is the step size, tuning the magnitude of each update.

- **Incorporating Annealing Noise Schedule**

  To improve convergence, the system employs an annealing noise schedule that adapts the noise variance σkσk over the iterations. The update rule, which now explicitly accounts for the noise level, is given by:

  $$y^{(k+1)} = y^{(k)} + \eta \left(g_\phi(y^{(k)}; x; \sigma_k) - y^{(k)} \right) \tag{9}$$

  where the σ_k is the noise variance, Initially set to a relatively high value, the noise variance is gradually decreases with the iteractions. This approach helps the model correct large deviations in the early stages and fine-tune the estimate in later iterations.

  This approach works as an adaptive correction method: higher noise levels assist in navigating larger errors by providing broader corrective signals, whereas lower noise levels enable precise adjustments as the solution nears equilibrium.

- **Convergence Criterion**

  The equilibrium state y^* is reached when the corrective vector vanishes:

  $$v(y^*; x) = g_\phi(y^*; x) - y \tag{10}$$

  Equivalently, convergence is defined by the norm of the corrective vector field falling below a small threshold ϵ:

  $$\| \, v(y^{(k)}; x) \, \|_2 < \epsilon \tag{11}$$

  This threshold-based criterion ensures that the updates become negligibly small, indicating that $y^{(k)}$is essentially at an equilibrium state.

- **Clipping of Updates**

  Clipping of the vector field updates is employed to maintain stability in the refinement dynamics and prevent the algorithm from overshooting the local basin of attraction. This means that if the magnitude of the update $\eta \, v(y^{(k)}; x)$ exceeds a predetermined threshold, the update is scaled back:

  $$\Delta y^{(k)} = \text{clip} \left(\eta \, v(y^{(k)}; x), \text{min} = -c, \text{max} = c \right) \tag{12}$$

  where c is the clipping constant. Clipping ensures that no single update is too large, preserving the local convergence properties and mitigating the risk of diverging from the equilibrium basin.

Training Methodology

Before training the model,standard scaling was applied to different sets. The training stages correspond to two independent stages:

- **Direct Predictor Training:**

  Optimized by minimizing the direct prediction loss $\mathcal{L}_{pred}$ (MSE loss + L_2 regularization).

  Uses an MLP with two hidden layers, ReLU activations, and a regularization term (L_2) with $\lambda_{reg} = 10^{-5}$.

  The model architecture is relatively compact, with 4217 parameters, making it a lightweight direct predictor.

- **CDAE Training**:

  Optimized independently to minimize the denoising reconstruction loss $\mathcal{L}_{CDAE}$ (MSE loss).

  Training employs a noise schedule with 10 equally spaced noise levels ranging from 0.3 to 10^{-4}, ensuring the model is exposed to a progressive spectrum of corruption.

  All noise levels contribute equally to the loss (each with a weight of 1), promoting balanced learning across the noise spectrum.

- **Optimization details**

  Optimizer: Adam with a learning rate of 10^{-3}

  Dataset: Training set: 2550, Validation set: 135, Test set: 315, Batch size: 32

CDAE Model Implementation

The CDAE model includes several key components:

- **Noise Embedding**

  A dedicated sub-network processes the scalar noise into a higher-dimensional representation using two linear layers interleaved with a ReLu activation. This design improves the model's sensitivity to the noise scale.

- **Input Concatenation**

  The input for the encoder is formed by concatenating the input x, the noisy output $\tilde{y}$ and the noise embedding. This enables the model to condition the encoding on both the observed corrupted output and the associated noise level.

- **Encoder and Decoder**

The encoder consists of two linear layers followed by ReLU activations. The decoder re-integrates the input x and the noise embedding with the latent vector, through two linear layers interleaved with a ReLU. In this way, we aim to reconstruct the clean output by exploiting the learned latent features and the original conditioning information.

- **Forward Pass**:

  The forward pass is given by embedding the noise level, concatenating the inputs, encoding to a latent space, and then decoding back to a reconstruction of the desired output.

The model has 23548 parameters in total.

Results

1. Performance of the Direct Predictor

- **Metric**: RMSE (Root Mean Squared Error)
- **Result**: The direct predictor achieved an RMSE of **0.0345**.

This serves as the baseline for assessing the impact of subsequent refinement.

2. CDAE Refinement Results

- **Refinement Process**: Iterative updates using the learned corrective vector field from the CDAE.
- **Iterative Parameters**:
  - Number of iterations per noise level, $K = 700$
  - Step-size (learning rate) for refinement, $\eta = 10^{-3}$
- **Metric**: RMSE of the refined predictions
- **Result**: RMSE was significantly reduced to **0.0168**.
- **Gain**:
  The iterative refinement process achieved a **51% reduction** in RMSE compared to the direct predictor, illustrating the effectiveness of the CDAE refinement in driving the predictions closer to equilibrium.

3. Comparison with a Larger MLP Model

- **Model**: A comparable MLP model with increased capacity.
- **Architecture**:
  - Two hidden layers interleaved by ReLU activation functions
  - Number of parameters: 31,517
- **Metric**: RMSQ (Root Mean Squared Error for the test set)
- **Result**:
  - RMSQ Test Loss: **0.0341**

Despite having substantially more parameters, the larger MLP model did not achieve the same level of performance as the CDAE refinement approach. This comparison emphasizes that including the denoising iterative process and noise-augmented training leads to better generalization and lower error than simply increasing model size.

Conclusion

The results demonstrate that the CDAE surrogate model markedly enhances both the accuracy and reliability of predicting equilibrium states in deterministic plasma-surface interaction scenarios, outperforming traditional direct predictors. Several factors contribute to this improvement:

1. **Learning Interdependencies**:
   The CDAE is designed to capture and encode the complex interdependencies between the input variables x and the output y.The model effectively learns a richer representation of the underlying relationships by conditioning the denoising process on both the input and the noisy output. This leads to more a robust and accurate reconstruction of equilibrium states.
2. **Scalability with Data and Output Dimensions**:
   Due to its ability to understand the intrinsic relationships between different components, the CDAE is expected to scale well with larger datasets and higher-dimensional outputs. As more training samples become available, or as the complexity of the interaction increases, the model's performance can further improve, potentially outperforming models that do not explicitly model these interdependencies.
3. **Data Intensity and Architectural Specificity**:
   One challenge with the CDAE architecture is its data-intensive nature. Achieving optimal performance requires a sufficiently large and representative training set. Additionally, the specifics of the architecture, such as the design of the progressive conditional denoising autoencoder, play a critical role in determining the model's performance. This means fine-tuning and possibly increasing the training set are essential for extracting

the best results.

4. **Dependence on Initial Predictions**:
   The CDAE relies on an initial guess provided by a direct predictor. The refinement process improves the prediction iteratively, but its success is closely tied to the quality of the initial estimate. If the initial guess falls outside the local attractor basin of the correct equilibrium state, the refinement process may converge to an incorrect solution. This sensitivity underscores the importance of having a good base predictor to ensure the effectiveness of the iterative refinement.
5. **Comparison with Score-Based and Diffusion Models**:
   Unlike traditional methods, such as Langevin dynamics or forward processes commonly used in score-based generative models and diffusion models, designed to sample from a modeled distribution—your approach aims to converge to a specific equilibrium state. In diffusion models, guidance during inference (such as classifier-based guidance) can help steer the sampling process, which is not a factor in your deterministic scenario. Here, the focus is on refining a given initial guess rather than generating random samples, which highlights a significant difference between the two methodologies.

References

Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P. A. (2008). Extracting and Composing Robust Features with Denoising Autoencoders. In Proceedings of the 25th International Conference on Machine Learning (ICML '08) (pp. 1096–1103).

Ho, J., Jain, A., & Abbeel, P. (2020). Denoising Diffusion Probabilistic Models. In Advances in Neural Information Processing Systems (NeurIPS '20).

Song, Y. & Ermon, S. (2019). *Generative Modeling by Estimating Gradients of the Data Distribution.* Advances in Neural Information Processing Systems (NeurIPS).

Belanger, D., & McCallum, A. (2016). *Structured Prediction Energy Networks*.

Valente, Matilde, et al. "Physics-consistent machine learning: Output projection onto physical manifolds." *arXiv preprint*arXiv:2502.15755 (2025).

Appendix

The implementation details are presented on the GitHub repo [https://github.com/joseAf28/PlasmaCDAE]

In the following sections, we introduce different architectures that have also been explored, but whose results do not present a significant gain when compared to the simple direct predictor.

Autoregressive Model

We explored an alternative autoregressive model characterized by predicting output components sequentially:

- **Autoregressive Decomposition**: Each component y_i of the output y is predicted sequentially, conditioned on inputs x and all previously predicted components:

$$\hat{y}_i = h_i([\phi(x), \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_{i-1}]) \tag{13}$$

  Shared Backbone Network: Input parameters x are first transformed by a shared neural backbone network $\phi(x)$ into a common latent space. Each sequential prediction is then performed by specific subnetworks $h_i(\cdot)$.

- **Teacher Forcing with Cosine Scheduling**: During training, ground truth components y_j and predicted components $\hat{y}_j$ are mixed according to a cosine annealing schedule:

$$y_j^{input} = \begin{cases} y_j^{true}, & \text{with probability } p(t) \\ \hat{y}_j, & \text{with probability } 1 - p(t) \end{cases} \tag{14}$$

  where the scheduling function $p(t)$ is defined by:

$$p(t) = p_{\min} + \frac{1}{2}(p_{\max} - p_{\min}) \left(1 + \cos\left(\pi \frac{t}{T}\right)\right) \tag{15}$$

  where $p_{\min}$ and $p_{\max}$ specify the minimum and maximum teacher forcing probabilities and t denotes the current training step, and T is the total number of training steps.

  This scheduling gradually shifts the network from relying heavily on the ground truth (at the beginning of training) to relying more on its predictions as training progresses.

- **Residual Correction Network**: An additional correction network computes residual corrections Δy, refining the base autoregressive predictions:

$$\hat{y}_{\text{final}} = \hat{y}_{\text{base}} + \Delta y \tag{16}$$

  This correction step helps mitigate errors that accumulate over the sequence and allows the network to adjust the final output further. However, it remains challenging to thoroughly recover from errors introduced in early sequence predictions.

Performance and Limitations

- **Performance**:
  In our experiments, the autoregressive model reached an RMSE of approximately 0.046 on the test set. The total model capacity was 25843 parameters.

- **Order Dependency**:
  The autoregressive approach requires selecting an ordering to predict y. In our experiments, we used an arbitrary order based on the order of the dataset columns. Note that there are $17!$ possible orderings for the 17 output components. Based on physical insights into the system, an informed ordering might improve performance by better capturing the interdependencies between components. However, even with a carefully chosen order, the model is inherently limited by its sequential structure.
- **Interdependency Limitation**:
  The sequential nature restricts the model's capacity to capture simultaneous and complex interdependencies between the y and x components. In contrast, the CDAE model processes the joint information in a more structured and integrated way.
- **Error Propagation**:
  The autoregressive model suffers from error propagation despite using teacher forcing with a cosine schedule during training. If an error is made early in the sequence, it can propagate through subsequent predictions, making it difficult for the model to recover from previous mistakes.

Comparison to the Progressive CDAE Model

- The progressive CDAE model has a more integrated approach, capturing the joint distribution of x and y and using iterative refinement to correct its predictions. It overcomes several key limitations of the autoregressive approach:
  - **Parallel Recovery of Interdependencies**:
    Instead of processing components sequentially, the CDAE model can leverage simultaneous information from all components.
  - **Resilience to Initial Errors**:
    The iterative refinement in the CDAE reduces sensitivity to the initial guess, provided it is within the basin of attraction. In contrast, the autoregressive model has inherent difficulties recovering from early errors.

Latent Space–Based Model with an Autoencoder

This formulation leverages a conditional denoising autoencoder (CDAE) to learn a compact latent representation that captures the intrinsic manifold of the system's equilibrium states. The main components and training stages are outlined below:

- **Autoencoder Framework**:
  The model is built as a CDAE comprising an encoder and a decoder:

- **Encoder**: Maps the combined inputs ,original output y (or its noisy variant) and the conditioning input x, into a latent space z.
- **Decoder**: Reconstructs y from the latent representation and the input x.

- **Noise Perturbation**:
  A noise model perturbs the output before encoding (i.e., $y\prime = y + \varepsilon$ with $\varepsilon \sim N(0, \sigma^2)$; the network is trained to reconstruct the original y from this corrupted version.
- **Mapping Network**:
  A separate network is trained to predict the latent space directly from the inputs xx. This network is then combined with the decoder to form a final predictor:

$$\hat{y} = D(x, F(x)) \tag{17}$$

  where $F(x)$ approximates the latent vector z learned by the encoder.

Training Stages:
The training is divided into three main steps:

1. **Stage 1 – Autoencoder Training**:
   The CDAE is trained using a mean squared error (MSE) loss to capture the latent manifold of the equilibrium states. This stage allows the model to learn a robust joint representation of (x, y) (or their noisy versions).
2. **Stage 2 – Mapping Network Training**:
   With the CDAE's encoder-decoder weights frozen, the mapping network $F(x)$ is trained to predict the latent space from the input xx, effectively aligning the predicted latent representation with that captured by the encoder.
3. **Stage 3 – Fine-Tuning the Final Predictor**:
   In the final step, the mapping network and the decoder are jointly fine-tuned using a loss function that compares the decoded output $\hat{y}$ against the ground truth y. Gradually, clipping is applied to stabilize the training.

Performance and Limitations:

In a proof-of-concept experiment using a previously presented dataset, the latent space–based model achieved an RMSE test loss of approximately 0.03636. The latent space has a low dimensionality ($\mid z \mid = 4$), ensuring that the model captures a compact representation. The model comprises the mapping network (which predicts the latent space from x) and the decoder (which predicts y from z), consisting of 5821 parameters.

Compared to the direct predictor—which achieved a lower RMSE—the gain provided by the latent space formulation is low (around 5%). This marginal improvement suggests that the increased complexity introduced by the latent space approach is poorly justified for our system. One potential reason for this limited gain is that the output space is only 17

dimensions, which is not very high-dimensional. When the dimensionality is relatively low, learning the output directly from the input is already manageable, and the additional step of constructing a latent representation does not confer a significant advantage.

Moreover, these results indicate that learning a meaningful and compact representation of the output space from the input is as challenging as learning the output directly. In this approach, we had hoped that the autoencoder would distill the output into a latent space that would be easier to predict from the input. However, the experimental evidence does not support this hypothesis; the latent space did not significantly simplify the learning task compared to the direct mapping.