Draft version December 30, 2025
Typeset using LaTeX **twocolumn** style in AASTeX7.0.1

The problem: ML models hate extremes! (They're biased!)

Why Machine Learning Models Systematically Underestimate Extreme Values II: How to Fix It with LatentNN

The solution: LatentNN to the rescue!

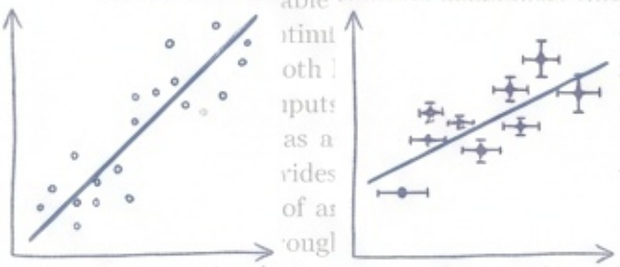
Yuan-Sen Ting[1,2,3]

[1]*Department of Astronomy, The Ohio State University, Columbus, OH 43210, USA*
[2]*Center for Cosmology and AstroParticle Physics (CCAPP), The Ohio State University, Columbus, OH 43210, USA*
[3]*Max-Planck-Institut für Astronomie, Königstuhl 17, D-69117 Heidelberg, Germany*

ABSTRACT

Attenuation bias—the systematic underestimation of regression coefficients when predictors contain measurement errors—affects nearly all astronomical data-driven models. For linear regression, this problem was solved by treating the true input values as latent variables to be estimated alongside model parameters. In this paper, I treat the true input values as latent variables to address the same attenuation bias and that [illegible] latent variable solution immediately [illegible] directly to neural networks. We introduce LatentNN, a method [illegible] optimizing [illegible] and latent input values by maximizing the joint likelihood [illegible] both [illegible] demonstrate [illegible] inputs [illegible]s, and stellar [illegible] as a [illegible] to-noise ratios [illegible] rides [illegible] ed neural networks [illegible] signal-to-noise regime of astronomical [illegible] bias correction is most effective when measurement errors [illegible] rough[illegible] ta range; in the regime of very low signal-to-noise and few [illegible] native features [illegible] https://github.com/tingyuansen/LatentNN.

Basically, noisy inputs make models lazy and squish their predictions towards the average. This is ATTENUATION BIAS.



True Relationship

Standard ML fit (Attenuated, boo!)

LatentNN says: "Hold on, those noisy inputs aren't the real story. Let me figure out what the *true* values probably were while I'm learning!"

Perfect for noisy data, like a lot of astronomy!

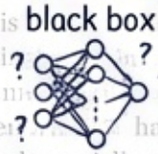
Keywords: methods: statistical — methods: data analysis — techniques: spectroscopic — stars: fundamental parameters

1. INTRODUCTION

Modern astronomy increasingly relies on neural networks for inference (Cranmer et al. 2020; Huertas-Company & Lanusse 2023; Ting 2025a). Data-driven approaches now routinely map observed quantities to physical parameters across a wide range of applications, from stellar parameters derived from spectra (e.g., Ness et al. 2015; Fabbro et al. 2018; Leung & Bovy 2019; Ting et al. 2019) to photometric redshifts from imaging (Hoyle 2016; Li et al. 2022a; Lin et al. 2022; Zhou et al. 2022) to steliar properties from time series light curves (Pasquet et al. 2019; Hon et al. 2021).

Astronomy has BIG data now, so we need ML to make sense of it all.

With the rise of neural network approaches, however, input uncertainties are often ignored or left unmodeled. This is [illegible] design choice rather than an oversight. Classical [illegible] learning [illegible] with [illegible] in input [illegible] framework [illegible] handling [illegible] network modeling, by contrast, arose in domains like computer vision (Simonyan & Zisserman 2014; Szegedy et al. 2014; He et al. 2016) and natural language processing (Devlin et al. 2018; Brown et al. 2020; OpenAI et al. 2023; Touvron et al. 2023) where input noise is negligible, and its standard architectures have little mechanism for directly [illegible] uncertainties. When applied to [illegible] application can lead [illegible] to diagnose (Kelly [illegible]



Most ML models are like: "La la la, I can't see your error bars!"

Astronomy operates in a regime that differs from most machine learning applications. Astronomical observations often have signal-to-noise ratios (SNR) of order 10, yet we seek inference precision at the percent level or better. This tension between noisy inputs and stringent precision requirements creates an unique challenge for neural network inference.

This phenomenon is known as attenuation bias (Spearman 1904; Frost & Thompson 2002; Kelly 2007). In Paper I of this series (Ting 2025b), we demonstrated this effect in detail for linear regression and explored its implications for astronomical data analysis. Attenuation bias manifests as a compression of the predicted dynamic range: high values are consistently predicted [illegible] The effect depends on the ratio of input measurement error

compression of the predicted dynamic range

High values predicted too low, low values too high.... it's all squashed to the middle!

to intrinsic signal range, and it is independent of sample size. As a rule of thumb, attenuation bias becomes important at input SNR of order 10, especially at low-dimensional problems. The exact magnitude depends on the dimensionality and correlation structure of the inputs, but the bias is never zero when input uncertainties are present. We refer interested readers to Paper I for detailed derivations and further discussion of attenuation bias in the context of linear regression.

The consequences of attenuation bias are not merely increased scatter in predictions. When we train a regression model to predict some quantity from noisy inputs, treating the observed inputs as exact is a reasonable approximation only when measurement errors are small compared to the intrinsic variation in the data. When measurement errors constitute even a modest fraction of the signal, a systematic compression of the predicted dynamic range emerges—high values are underestimated and low values are overestimated—a bias that cannot be removed by collecting more data or by using more sophisticated models. It is a fundamental consequence of treating noisy measurements as if they were exact.

Especially bad when data is noisy (low SNR)!

systematic compression of the predicted dynamic range.

This is the 'squashing' effect we saw on the last page!

Keywords: methods: ... fundamental parameters

1. INTRODUCTION

Modern astronomy increasingly relies on neural networks for inference (Cranmer et al. 2020; Huertas-Company & Lanusse 2023; Ting 2025a). Data-driven approaches now routinely map observed quantities to physical parameters across a wide range of applications, ... g., Ness et al. 2015; Fabbro et al. 2018; ... Ting et al. 2019) to photometric ... (Hoyle 2016; Li et al. 2022a; ... 2022) to stellar properties ... (Pasquet et al. 2019; Hon et al. ...

treat the true input values as latent variables.

The clever trick from old-school stats: pretend the true values are hidden variables we can solve for.

With the rise of neural network approaches, however, ... left unmodeled ... extending ... plies ... limitations, ... Bayesian modeling, ... Section 8.

extends this idea to neural networks.

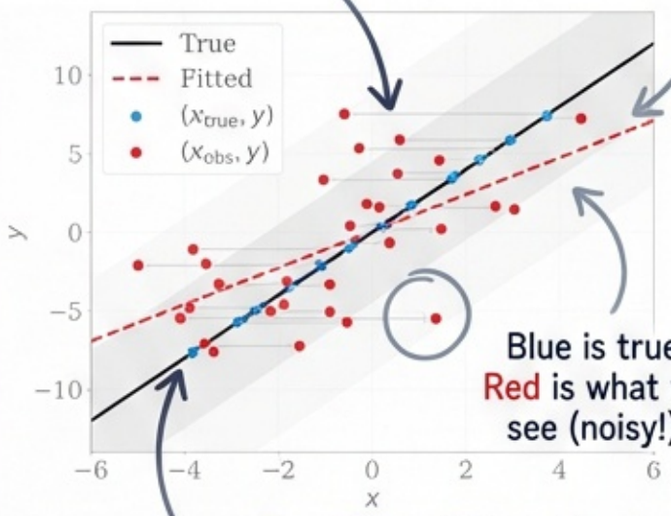
And now we're bringing this powerful idea to modern deep learning!

2. ATTENUATION BIAS

To understand attenuation bias, consider a simple regression problem where we want to learn the relation-

The TRUE relationship we want to find.

The BIASED fit from standard ML.



Blue is true, Red is what we see (noisy!).

See? The fitted line is too shallow! It's been SQUASHED towards the middle.

Figure 1. Schematic ... circles show true ... line); red squares ... adding measurement ... indicate horisontal displacement ... positions. The tilted grey shaded ... (darker) and $\pm 2\sigma_x$ (lighter) uncertainty regions around the true relationship. A linear fit to the observed data (red dashed line) yields a shallower slope, demonstrating how input meaeurement ... error. If these errors are unbiased with variance σ_x^2, then $\delta_x \sim \mathcal{N}(0, \sigma_x^2)$. The naive approach in machine learning is to train on the observed pairs ... x_{obs} as if it were the true value. This ... ous simplification leads to systematic ...

This is ATTENUATION BIAS in action. The noise makes the model underestimate the true slope.

Measurement errors in x artificially ... served variance. If the true values have ... $\text{Var}(x_{\text{true}})$, representing the intrinsic spread or dynamic range of the data, then the observed variance is

$$\text{Var}(x_{\text{obs}}) = \sigma^2_{\text{range}} + \sigma^2_x. \quad (1)$$

Basically, noise makes the data spread out more horizontally.

This stretches the data horizontally without a corresponding vertical stretch, because the measurement errors in x do not affect y. The observed distribution appears wider in the x-direction than the true distribution, while the y-extent remains unchanged. Any regression method treating x_{obs} as exact will therefore fit a slope that is systematically shallower than the true relationship. This weakening of the apparent relationship is why attenuation bias is also known as regression dilution.

regression dilution

A.k.a. "watering down" the relationship.

This effect can be visualized by plotting data points and their regression line. Figure 1 illustrates this with a simple example: true data points (blue) following a linear relationship $y = 2x$ are scattered horizontally by measurement error to their observed positions (red), with arrows indicating the displacement. Grey tilted … gions ar… line fit to the observed … shallower slope than the true underlying relationship.

THE "SQUASHING" FACTOR
(MATH EDITION)

For … $= \beta x$, wh… …fect prec… The ordinary least squares estimator $\hat{\beta}$ has expected value

This number tells us how much the line gets flattened (always < 1)

$$\mathbb{E}[\hat{\beta}] = \beta \cdot \lambda_\beta, \quad \text{where} \quad \lambda_\beta = \frac{1}{1 + (\sigma_x/\sigma_{range})^2}. \quad (2)$$

Noise level

True signal range

The quantity λ_β … attenuation factor. It depends only on … intrinsic signal range, σ_x/σ_{range}. When meas… errors are small compared to the signal range, λ_β is close to unity and the bias is negligible. When measurement errors are comparable to the signal range, λ_β can be much less than unity, indicating large bias. Since $\lambda_\beta < 1$, predic-tion… high values ar… are predicted too high. In the extreme case where measurement errors dominate ($\sigma_x \gg \sigma_{range}$), λ_β approaches zero and the model predicts the mean for all inputs, obscuring the true relationship.

It's the ratio of NOISE to SIGNAL that dictates the damage.

Consider some representative cases in the one-dimensional setting:

Real-world examples of the damage:

- At $\sigma_x/\sigma_{range} = 0.1$ ($\sigma_{range}/\sigma_x = 10$), the attenuation factor is $\lambda_\beta \approx 0.90$, corresponding to 1% bias. — 1% bias (Not bad)
- At $\sigma_x/\sigma_{range} = 0.33$ (SNR ≈ 3 … factor is $\lambda_\beta \approx 0.90$, correspondi… — 10% bias (Getting worried...)
- At $\sigma_x/\sigma_{range} = 1.0$ … factor is $\lambda_\beta = 0.50$, — 50% BIAS! Model is basically guessing the mean!

At SNR of 10 or below, a regime com… cal observations, the bias reaches percent level or more.

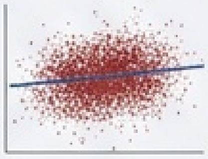
Common in astro! →

Several properties of this bias are worth emphasizing. First, the bias depends only on the ratio σ_x/σ_{range}, not on the absolute values of either quantity. Second, it is independent of sample size n and cannot be resolved by collecting more data. Even with infinite training samples, the bias remains. Third, the magnitude of attenuation bias depends only on errors in x, not errors in y. Errors in y add scatter to the predictions but do not systematically bias them, because positive and negative errors average out over the dataset. In other words, obtaining better or more accurate labels does not solve the problem.[4]

This asymmetry between x and y arises because they play fundamentally different roles in regression: we predict y given x, not the reverse. Errors in x change the question being asked—we are predicting from a corrupted version of the input—which systematically weakens the apparent relationship. However, as we will see in Section 4, while σ_y does not affect the bias itself, it does play a role in correcting the bias: the Deming regression solution (Deming 1943; Golub & Van Loan 1980; Ting 2025c) and its neural network generalization require specifying both σ_x and σ_y to properly weight the loss terms.

Why this bias is excessively annoying:

- ☒ Getting MORE data doesn't fix it. (It's a *bias*, not variance!)
- ☒ It's caused by errors in the INPUTS (x), not the labels (y).



So, linear math is neat, but what about...

3. ATTENUATION BIAS IN NEURAL NETWORKS

rived analytically. Paper I further argued that the same bias extends to polynomial and nonlinear models. However, for nonlinear models or high-dimensional inputs, the attenuation factor can no longer be expressed as a simple function of σ_x and σ_{range}. The relationship between measurement error and attenuation becomes more complex, depending on the specific functional form.

This complexity makes it difficult to predict and therefore analytically correct the bias in closed form for general machine learning models. Here we first demonstrate chesbeanes tinesoeeserate mefud cerenes.

This complexity makes it difficult to predict and there-analytically correct the bias in closed form for general machine learning models. …e first demonstrate empirically that multi-layer perceptrons (MLP) exhibit the same qualitative layer perceptrons (MLPs) exhibit the same qualitative behavior. MLPs, also known as fully connected neural networks (Cybenko 1988; Hornik et al. 1989), are the simplest neural network architecture: each neuron in one layer is connected to every neuron in the next layer, with nonlinear activation functions between layers. Despite their simplicity, MLPs remain widely used in astronomy (e.g., Odewahn et al. 1992; Storrie-Lombard et al. 1992; Collister & Lahav

The math gets too hard here. No simple formula to save us.

But the problem is still there! Our fancy deep learning models get lazy too.

OK, let's test this on Neural Networks! Do they get lazy too?

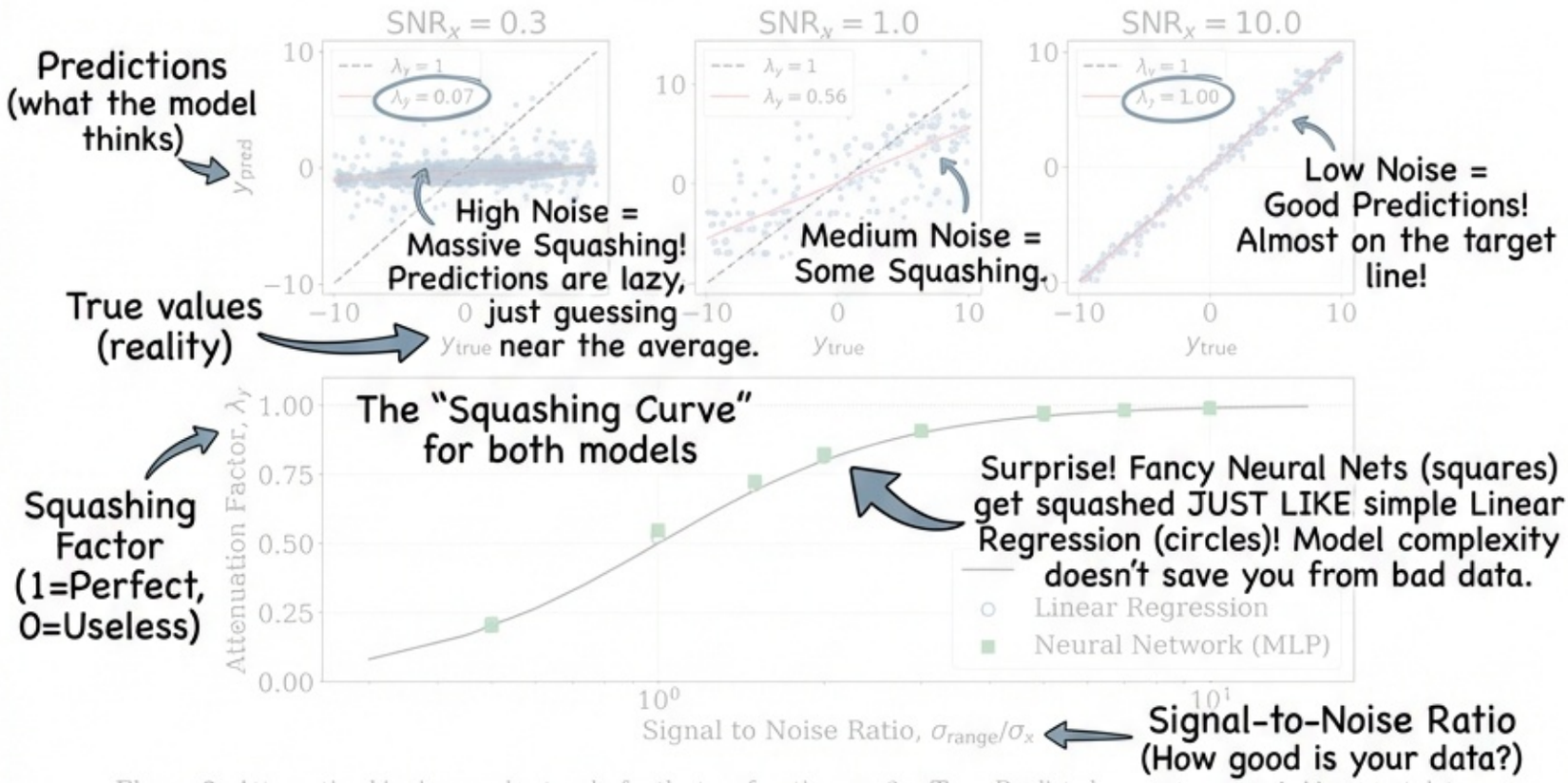


Figure 2. Attenuation bias in neural networks for the true function $y = 2x$. **Top:** Predicted versus true y on held-out test data for an MLP (2 hidden layers, 64 units) at three SNR$_x$ values. At SNR$_x$ = 0.3, predictions collapse to $\lambda_y \approx 0.1$; at SNR$_x$ = 1, $\lambda_y \simeq 0.5$; at SNR$_x$ = 3, $\lambda_y \approx 0.9$. **Bottom:** Attenuation factor λ_y versus SNR$_x$ for linear regression (blue open circles) and neural network (green filled squares). Both follow the theoretical curve $\lambda_y = 1/(1 + \text{SNR}_x^{-3})$ (black solid line), demonstrating that model complexity does not protect against attenuation bias.

2004; Guo et al. 2020; Wong et al. 2020; Wang et al. 2021; Li et al. 2022b; Vynatheya et al. 2022; Wincki & Kochanek 2024) and serve as the foundation for more complex architectures.

Here's our experiment recipe: take a simple relationship, then corrupt the inputs with noise.

To test whether neural networks suffer from attenuation bias, we conduct numerical experiments following the setup of Paper I. We consider the true function $y = 2x$, chosen for direct comparison with the linear theory, and sample $N_{\text{train}} = 1000$ training values and $N_{\text{test}} = 200$ test values of x_{true} uniformly from $[-5, 5]$. We add Gaussian noise to obtain $x_{\text{obs}} = x_{\text{true}} + \delta_x$ where $\delta_x \sim \mathcal{N}(0, \sigma_x^2)$, and vary σ_x to test different SNR regimes. We define $\text{SNR}_x \equiv \sigma_{\text{range}}/\sigma_x$ as the signal-to-noise ratio in the input.

We train two models on each dataset. The first is ordinary least squares linear regression. The second is an MLP with 2 hidden layers of 64 units each and ReLU activations, trained for 20000 epochs using the Adam optimizer with learning rate 0.03.

Since NNs are complex black boxes, we measure the squashing by slope of predictions vs reality.

For neural networks, we cannot directly measure a single coefficient β as we did for linear regression in Section 2. The MLP used here has over 4000 parameters distributed across multiple layers, rather than a single interpretable slope. Instead, we quantify attenuation through the predictions themselves. We define λ_y as the slope of the regression of y_{pred} against y_{true}. If a model suffers from attenuation bias, its predictions will be compressed toward the mean: when we plot y_{pred} versus y_{true}, the slope will be less than unity. A value of $\lambda_y = 1$ indicates no attenuation, while $\lambda_y < 1$ indicates that predictions are systematically pulled toward the sample mean.

As discussed in Paper I, for linear regression in one dimension with true relationship $y = \beta x$, these two measures of attenuation are equivalent. The predicted value is $\hat{y} = \hat{\beta} x_{\text{obs}}$, and when we regress predictions against true values, the attenuation in the slope estimate $\hat{\beta}$ translates directly to attenuation in the predictions. In this case, $\lambda_y = \lambda_\beta = 1/(1+(\sigma_x/\sigma_{\text{range}})^2)$. For nonlinear models or multivariate inputs, λ_y remains a useful empirical measure of how much the model compresses the dynamic range of predictions.

Figure 2 shows the results. The top row shows MLP predictions on held-out test data at three specific SNR$_x$ values: at SNR$_x$ = 0.3 (severe noise), predictions are compressed to $\lambda_y \approx 0.1$; at SNR$_x$ = 1 (noise comparable to signal range), $\lambda_y \approx 0.5$; and at SNR$_x$ = 3, $\lambda_y \approx 0.9$. The bottom panel shows that both linear re-

Since NNs are complex black boxes, we measure the squashing by checking the slope of predictions vs reality.

So, complex models don't save us. What now? Enter our proposed solution: LATENTNN!

gression and the MLP follow the theoretical attenuation curve $\lambda_y = 1/(1 + (\sigma_x/\sigma_{\text{range}})^2)$ across the full range of SNR_x. For the linear true function $y = 2x$, the MLP effectively learns a linear mapping, so it follows the same theoretical curve as linear regression. The important point is that our MLP here, despite having thousands of parameters and the flexibility to fit arbitrary nonlinear functions, exhibits the same systematic bias as the much simpler linear regression. As we argued in Paper I, model complexity does not protect against attenuation bias.

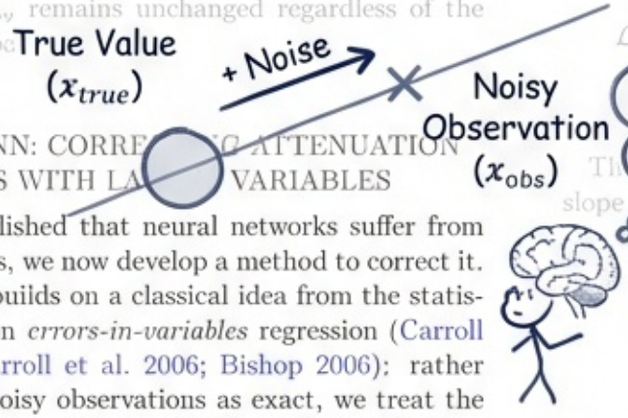
Regularization does not resolve the issue either. One might hope that L_2 regularization (weight decay), which penalizes large weights, could counteract the attenuation (Tikho

Quick detour: You might think "regularization" (a common trick to prevent overfitting) would help. Nope! It fixes overfitting, not this noise bias problem.

The Core Idea: Don't treat noisy data as "truth". Instead, treat the *true, unknown* values as hidden ("latent") variables that we need to estimate along with the model!

mismatch b

cannot correct. We verified this empirically by sweeping over a range of L_2 regularization strengths: the attenuation factor λ_y remains unchanged regardless of the regularization pa



4. LATENTNN: CORRE[illegible]G ATTENUATION BIAS WITH LA[illegible] VARIABLES

Having established that neural networks suffer from attenuation bias, we now develop a method to correct it. The approach builds on a classical idea from the statistics literature on *errors-in-variables* regression (Carroll et al. 1995; Carroll et al. 2006; Bishop 2006): rather than treating noisy observations as exact, we treat the unknown true values as latent variables to be estimated alongside the model parameters. For linear regression, this leads to Deming regression (Deming 1943). Here we show that the same principle extends naturally to neural networks, yielding a practical correction method we call LatentNN.

4.1. *The Latent Variable Solution in Linear Regression*

Standard regression treats x_{obs} as exact, which leads to bias. An alternative approach recognizes that x_{true} is an unknown quantity for each data point. Rather than treating the observed values as truth, we can treat the true values as latent variables to be estimated alongside the model parameters.

Consider observation i. We assume that the observed input $x_{\text{obs},i}$ is a noisy measurement of the true value $x_{\text{true},i}$, and the observed output $y_{\text{oba},i}$ is a noisy measurement of the true output $y_{\text{true},i}$, which follows the

linear relationship $y_{\text{true},i} = \beta x_{\text{true},i}$. Mathematically,

$$x_{obs,i} = x_{true,i} + \delta_{x,i}, \quad \delta_{x,i} \sim \mathcal{N}(0, \sigma_x^2), \tag{3}$$
$$y_{obs,i} = \beta x_{true,i} + \delta_{y,i}, \quad \delta_{y,i} \sim \mathcal{N}(0, \sigma_y^2). \tag{4}$$

Given these assumptions, the joint probability of observing both $x_{\text{obs},i}$ and $y_{\text{obs},i}$ given the true value $x_{\text{true},i}$ and the model parameter β is

Here serving x from a Ganssian distribution with mean μ and

Over all N observations, the full likelihood is

$$\mathcal{L}(\beta$$

(6)

This slope

for $2N$ observe

first glance this appears underdetermined. How-

$$x_{obs,i} = x_{true,i} + \delta_{x,i}, \quad \delta_{x,i} \sim \mathcal{N}(0, \sigma_x^2), \tag{3}$$
$$y_{obs,i} = \beta x_{true,i} + \delta_{y,i}, \quad \delta_{y,i} \sim \mathcal{N}(0, \sigma_y^2). \tag{4}$$

This is just writing down the problem mathematically: Observed = True + Noise.

This big equation is just asking: "Given the messy data we have, what's the most likely set of *true* values and the *true* line slope (β)?"

at the top level, the true values $x_{\text{true},i}$ form the intermediate level, and the actual measurements $(x_{\text{oba}}, y_{\text{oba}})$ sit at the bottom level. The model constrains the intermediate level, while the intermediate level generates the observations.

This seems hard (too many unknowns!), but it works because all the "true" points <u>must</u> fall on a single straight line. That's a huge constraint that makes it solvable!

The linear case (Deming regression) has an exact, albeit scary-looking, mathematical solution...

defining the error variance ratio $r = \sigma_y^2/\sigma_x^2$, the estimator is

$$\hat{\beta}_{\text{Deming}} = \frac{S_{yy} - rS_{xx} + \sqrt{(S_{yy} - rS_{xx})^2 + 4rS_{xy}^2}}{2S_{xy}}, \quad (7)$$

where $S_{xx} = \sum_i (x_{\text{obs},i} - \bar{x})^2$, $S_{yy} = \sum_i (y_{\text{obs},i} - \bar{y})^2$, and $S_{xy} = \sum_i (x_{\text{obs},i} - \bar{x})(y_{\text{obs},i} - \bar{y})$ are the sum of squared deviations and cross-products, with $\bar{x}$ and $\bar{y}$ denoting the sample means. In the limit $\sigma_x \to 0$ (equivalently $r \to \infty$), this reduces to the ordinary least squares estimator $\hat{\beta}_{\text{OLS}} = S_{xy}/S_{xx}$. When $\sigma_x > 0$, the Deming estimator yields a steeper slope than OLS, correcting for the attenuation (see also Ting 2025b).

The solution for the optimal $x_{\text{true},i}$ provides further intuition. For fixed β, the optimal estimate is a weighted average

$$x_{\text{true},i} = w_x \cdot x_{\text{obs},i} + w_y \cdot \frac{y_{\text{obs},i}}{\beta}, \quad (8)$$

where the weights are (see Appendix B for derivation)

$$w_x = \frac{r}{r + \beta^2}, \quad w_y = \frac{\beta^2}{r + \beta^2}. \quad (9)$$

The weights sum to unity. When σ_x is small compared to $\sigma_y/$[illegible] dominates and [illegible] is large (i.e., [illegible] we infer the true x from the y measurement and the model relationship. This optimal weighting is what allows the method to [illegible]

The intuition is beautiful: it's a smart WEIGHTED AVERAGE. Trust the measurement that is *less noisy*.

NOW, THE KEY INSIGHT: The "linear" part wasn't essential. The *hierarchical structure* was.

4.2. *The LatentNN Formulation*

The linear form of Deming regression was not essential—what mattered was the hierarchical structure where latent values generate observations and the model constrains the latent values. The linear form permits an analytic solution, but the underlying principle of joint optimization over model parameters and latent true values extends directly to neural networks.

We replace the linear model βx with a neural network $f_\theta(x)$, where θ represents all the network weights and biases. The hierarchical structure remains the same: the network parameters θ sit at the top level (analogous to β), the latent values $x_{\text{latent},i}$ form the intermediate level (analogous to $x_{\text{true},i}$), and the observations $(x_{\text{obs}}, y_{\text{obs}})$ sit at the bottom level. The likelihood has exactly the same form as in the linear case,

$$\mathcal{L}(\theta, \{x_{\text{latent},i}\}) = \prod_{i=1}^{N} \mathcal{N}(x_{\text{obs},i} | x_{\text{latent},i}, \sigma_{x,i}^2) \times \mathcal{N}(y_{\text{obs},i} | f_\theta(x_{\text{latent},i}), \sigma_{y,i}^2), \quad (10)$$

with $f_\theta(x_{\text{latent},i})$ replacing $\beta x_{\text{true},i}$. Here $\sigma_{x,i}$ and $\sigma_{y,i}$ denote the per-sample [illegible] across observations.

Taking the negative lo[illegible] absorbing constants, we obtain the loss function

So, we just swap the linear model (β_x) for a Neural Net ($f_\theta(x)$)!

$$\mathcal{L}(\theta, \{x_{\text{latent},i}\}) = \sum_{i=1}^{N} \frac{(y_{\text{obs},i} - f_\theta(x_{\text{latent},i}))^2}{\sigma_{y,i}^2} + \sum_{i=1}^{N} \frac{(x_{\text{obs},i} - x_{\text{latent},i})^2}{\sigma_{x,i}^2}. \quad (11)$$

Comparing to the Deming regression objective in Appendix B, we see the same structure: a prediction loss term weighted by $1/\sigma_{y,i}^2$ and a latent likelihood term [illegible] observations to improve prediction. This is the same [illegible] in the weight[illegible] $x_{\text{true},i}$ [illegible] measurement [illegible] but infer the true value from the model when the measurement is noisy. The uncertainties $\sigma_{x,i}$ and $\sigma_{y,i}$ must be specified for proper correction. [illegible] uncertainties σ_x [illegible] σ_y across all samples.

This leads to a new "loss function" to minimize, with the same two parts:

(1) Match observations

(2) Keep latents close to data

The optimization problem is to find

$$\hat{\theta}, \{\hat{x}_{\text{latent},i}\} = \underset{\theta, \{x_{\text{latent},i}\}}{\arg\min} \; \mathcal{L}(\theta, \{x_{\text{latent},i}\}). \quad (12)$$

For a dataset with N samples and p input dimensions, this involves optimizing over the network parameters θ plus $N \times p$ latent variables. The total number of parameters is thus much larger than in standard neural network training, scaling with the training set size.

The key difference from the linear case is that Equation 12 has no closed-form solution. In Deming regression, the linear structure allows us to solve for both $\hat{\beta}$ and $\{\hat{x}_{\text{true},i}\}$ analytically (see Appendix B). For a neural network f_θ, the nonlinear dependence on θ precludes an analytic solution, and we must resort to numerical optimization.

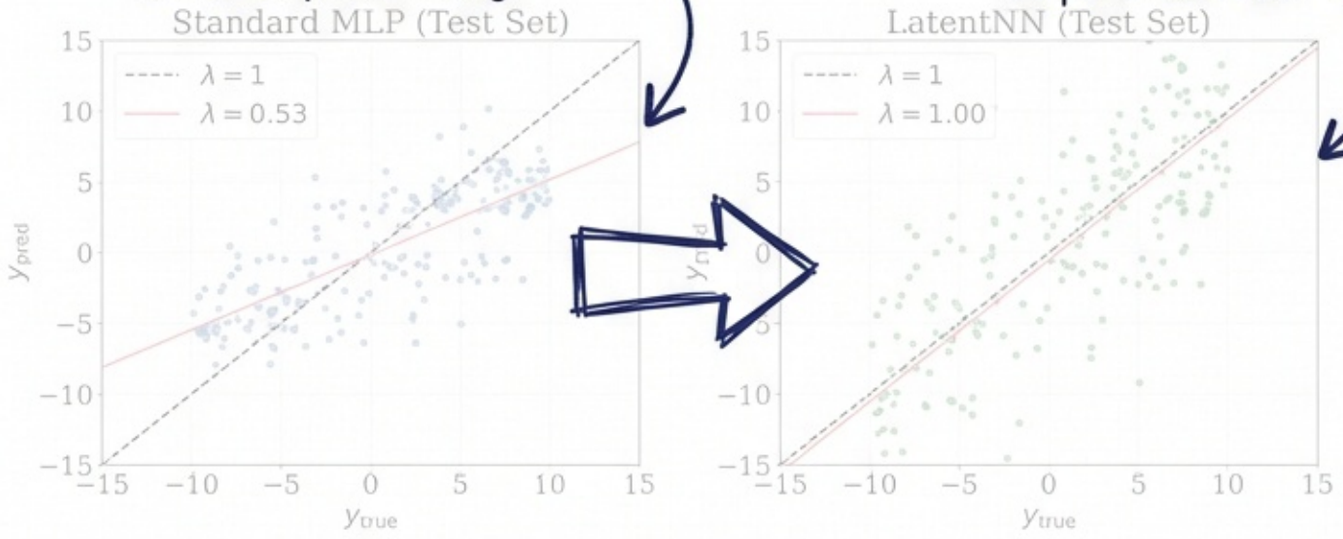
4.3. *Implementation*

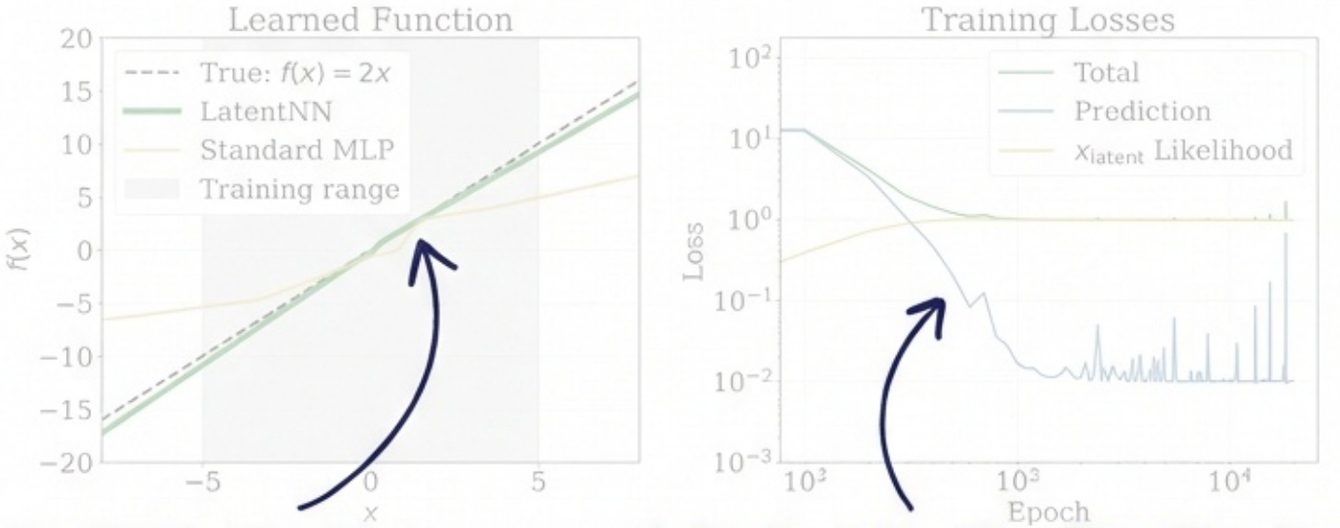
We solve Equation 12 using gradient descent, the standard optimization algorithm for neural networks. Let

THE CATCH: Unlike the linear case, there's no simple formula for the answer. We have to use numerical optimization (like gradient descent) to find it.

See? The standard Neural Net (MLP) is still SQUASHED by the noise! Slope is wrong.

But our LatentNN fixes it! The predictions line up with the truth! Slope is correct.





This plot shows the actual functions they learned. The true function is f(x)=2x (dashed). The standard one is too flat, but LatentNN nailed it!

This shows the training in action. The model is balancing two goals: fitting the data (Prediction loss ↓) and keeping latent values plausible (x_{latent} Likelihood ↑). It's a tug-of-war!

$\Phi = (\theta, \{x_{\text{latent},i}\})$ denote the full set of parameters to be optimised. At each iteration t, we update

$$\Phi^{(t+1)} = \Phi^{(t)} - \eta \nabla_{\Phi} \mathcal{L}, \tag{13}$$

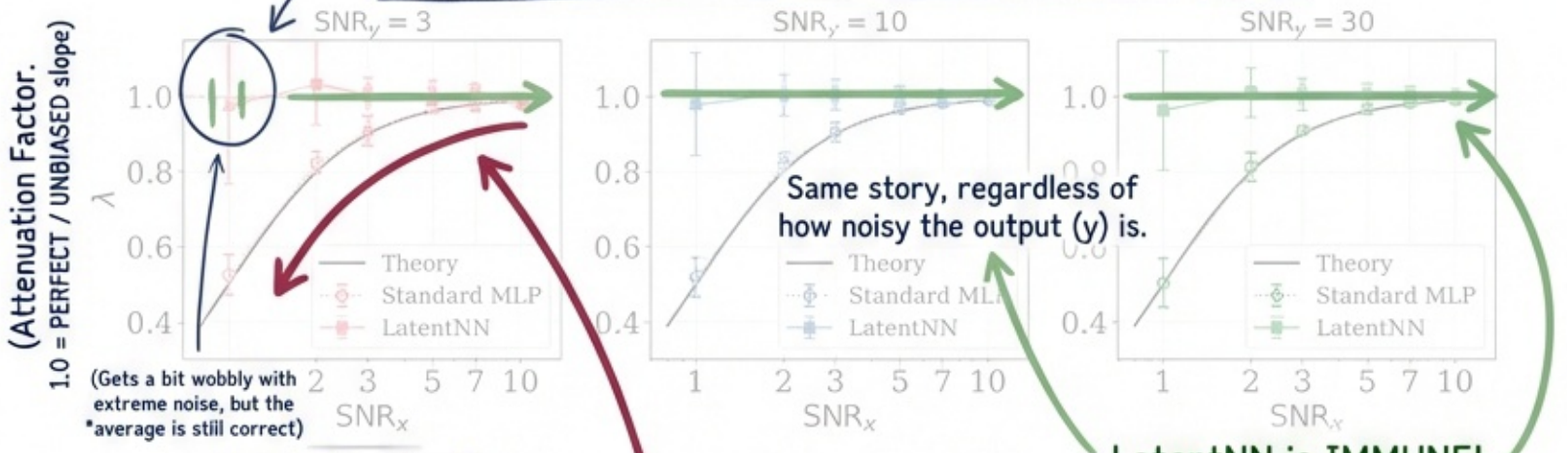
where η is the learning rate and $\nabla_{\Phi}\mathcal{L}$ is the gradient of the loss with respect to all parameters. This is the same update rule used in standard neural network training, but with Φ now including both the network weights θ and the latent inputs $\{x_{\text{latent},i}\}$. This represents a conceptual shift from typical neural network applications, where we optimize only over θ. Here, we treat the observed inputs as noisy measurements rather than ground truth, and optimize jointly over θ and the latent inputs.

In modern deep learning frameworks such as PyTorch or JAX, this is straightforward to implement. The la-

Here's that "numerical optimization" we mentioned! It's just standard gradient descent, but we update *everything* at once...

...by treating the latent values as just more parameters to learn! Clever trick.

Here's the SYSTEMATIC PROOF.
LatentNN vs. Standard MLP across ALL noise levels.



(Attenuation Factor.
1.0 = PERFECT / UNBIASED slope)

(Gets a bit wobbly with extreme noise, but the *average is still correct)

(Input Signal-to-Noise.
Left = VERY NOISY inputs)

Same story, regardless of how noisy the output (y) is.

LatentNN is IMMUNE!
It stays unbiased (λ≈1) even with crazy high input noise.

The Standard Net gets "squashed" by noise, exactly as theory predicts. It fails hard here! →

Figure 4. SNR$_x$ for three SNR$_y$ levels (3, 10, 30; different deviation seeds, using a three-fold data split (train/ hyperparameter selection. Standard MLP (open circles, dotted) follows the theoretical of SNR$_y$. LatentNN (filled squares, solid) maintains mean $\lambda_y \approx 1$ across all tested SN achieves unbiased predictions on average, though with increased run-to-run scatter at lower SNR$_x$ as the problem becomes increasingly ill-conditioned.

of network parameters, the yperparameter selection and the training tions where the network ove configurations rather than lo ping. Weight decay on the network parameters prevents this by penalizing large weights, encouraging smoother functions that generalize better. Weight decay is applied only to θ, not to the latent values x_{latent}—the latent values are already constrained by the latent likelihood term.

shows a representative comparison at SNR$_x$ = 1, using SNR$_y$ = 10 as in Section 3. At SNR$_x$ = 1, the measurement error equals the signal range ($\sigma_x = \sigma_{\text{range}}$)—an extreme regime where the noise is 100% of the data's dynamic range. The top panels show predicted versus true values on the held-out test set. The standard MLP shows $\lambda_y \approx 0.5$, demonstrat- LatentNN recovers $\lambda_y \approx 1$, The bottom-left panel shows the learned functions: the standard MLP learns an attenuated slope, while LatentNN recovers the true relationship $f(x) = 2x$.

The bottom-right panel shows the training dynamics for LatentNN. At initialization, $x_{\text{latent}} = x_{\text{obs}}$, so the x_{latent} likelihood term starts near zero. As training proceeds, the prediction loss decreases as the network learns the mapping, but the x_{latent} likelihood increases. This increase is expected and desirable: to achieve accurate predictions, the latent values must shift from the noisy values x_{true}, which in- The total loss decreases in prediction accuracy At convergence, the optimization has found the balance point where the latent values have moved sufficiently toward their true positions to enable unbiased prediction.

To systematically evaluate performance, we repeat each experiment S times with different random seeds, using the same three-fold data split (train/validation/test)

Time for a rigorous, fair test...

5. NUMERICAL EXPERIMENTS

We now validate the LatentNN approach on test cases where the true relationshlp and measurement errors are known, allowing direct comparison between estimated and true attenuation factors. We begin with the one-dimensional case then extend to multivariate inputs with correlated features.

3.1. *One-Dimensional Case*

We use the same data setup and network architecture as in Section 3: the true relationship $y = 2x_{\text{true}}$ with 1000 training samples, and an MLP with 2 hidden layers of 64 units trained for 20000 epochs. For each sample, we add a learnable parameter $x_{\text{latent},i}$ initialized to $x_{\text{obs},i}$.

We use a strict data split to prevent "cheating" (data leakage).

To ensure fair comparison with no data leakage, we use a three-fold data split: 1000 samples for training, 200 for validation (hyperparameter tuning), and 200 for final evaluation. The validation set is used exclusively for selecting hyperparameters—for LatentNN we maximize λ_y on the validation set, while for the standard MLP we minimize validation loss. Final performance metrics are computed on the held-out test set, which is never

CRITICAL DETAIL: They are tuned differently!
Standard MLP just optimizes for low *error*, while LatentNN is specifically tuned to get the *slope* (λ) correct on validation data.

The secret sauce: The two loss terms balance each other differently depending on the noise level.

ing three SNR_y values (3, 10, and 30). As expected from the theory in Section 2, varying SNR_y has little effect on attenuation—the bias depends primarily on input nolse. The standard MLP follows the theoretical attennation curve $\lambda_y = 1/(1 + (\sigma_x/\sigma_{\mathrm{range}})^2)$ regardless of SNR_y. LatentNN maintains $\lambda_y \approx 1$ across all tested SNR_x and SNR_y values. LatentNN provides robust correction even at $\mathrm{SNR}_x = 1$, though with increased run-to-run scatter at lower SNR_x values as the problem becomes increasingly ill-conditioned.

The mechanism of correction differs between SNR regimes. At high SNR where $\sigma_x/\sigma_{\mathrm{range}} \ll 1$, the x_{latent} likelihood term in Equation 11 dominates and the latent values remain close to the observations ($\sigma_{\mathrm{latent}} \approx x_{\mathrm{obs}}$). At moderate SNR ($\mathrm{SNR}_x \approx 2$–3), the two loss terms compete more evenly, and weight decay tuning becomes important for LatentNN. The standard MLP, by contrast, fails regardless of weight decay—it lacks the mechanism to correct attennation blas, and adjusting regularization only trades off between overfitting and underfitting without addressing the systematic bias.

LatentNN generalizes well to held-out test data. Weight decay regularization applied only to the network parameters θ prevents the network from overfitting to the specific denoised distributi… values. The test set performanc… ing set performance across all SNR va… that the learned function f_θ captures the true underlying relationship rather than attifacts … data.

5.2. *Multivariate Inputs*

Real astronomical applications involv… inputs. The LatentNN formulation gener… the latent variables $\mathbf{X}_{\mathrm{latent}}$ form an $N \times p$ matris, where p is the number of input features. The loss function from Equation 11 extends naturally to

$$\mathcal{L} = \sum_{i=1}^{N} \frac{(y_i - f_\theta(\mathbf{X}_{\mathrm{latent},i}))^2}{\sigma_{y,i}^2} + \sum_{i=1}^{N}\sum_{j=1}^{p} \frac{(X_{\mathrm{obs},ij} - X_{\mathrm{latent},ij})^2}{\sigma_{x,ij}^2}, \qquad (14)$$

where $\sigma_{y,i}$ is the output uncertainty for sample i and $\sigma_{x,ij}$ is the input uncertainty for sample i and dimension j. For simplicity, we assume homogeneous noise σ_x and σ_y across all samples and dimensions, which allows us to visualize results as a function of a single SNR_x.

As shown in Paper I, for uncorrelated features each dimension attenuates independently, recovering the one-dimensional result. We therefore focus on correlated features, which better represent astronomical data like stellar spectra where different pixels respond coherently to changes in stellar parameters. We generate $N_{\mathrm{train}} = 1000$ training samples where all features derive from a single latent variable: $x_j = a_j z$ where $z \sim \mathrm{Uniform}(-0.5, 0.5)$ and a_j varies linearly from 0.2 to 0.4 across dimensions. The true output is $y = \mathbf{x}^\top \mathbf{a}$, centered at zero. We fix $\mathrm{SNR}_y = 10$ as in the single-variate experiments. We use the same network architecture and training settings as Section 3, with the same three-fold data split (1000 train, 200 validation, 200 test) and weight decay grid search strategy. We test $p = 3, 10, 30$ dimensions at SNR_x values from 1 to 10, repeating each experiment 8 times.

Figure 5 shows the results. Paper I derived an analytic expression for the attennation factor in correlated linear regression: $\lambda_y = \lambda_\beta = \sum_j a_j^2/(1/\mathrm{SNR}_x^2 + \sum_j a_j^3)$. The gray curves show this theoretical prediction. Just as in the single-variate case, the standard MLP closely follows this linear regression theory curve, confirming that additional model complexity does not mitigate the fundamental attennation bias. With correlated features, attenuation is weaker than the $p = 1$ case because independent noise in different dimensions must conspire coherently to displace observations along the signal …idence that becomes increasingly im… With increasing p, the standard MLP …n factor is mitigated compared to smaller p, but remains significant at lower SNR.

Now for the REAL WORLD stuff: Multivariate Inputs!

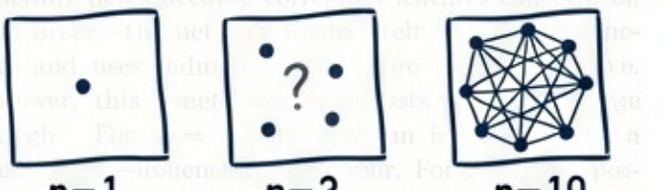
Term 1: Minimize prediction error on output (y).

Term 2: Constrain latent X to be plausible given noisy input.

The full multivariate loss function.

…results reveal a non-monotonic re… …ship between dimens… For $p \geq 10$, Laten… …ever, correction degrades … $\mathrm{SNR}_x = 1$ and $\lambda_y \approx 0.9$ at $\mathrm{SNR}_x = 2$. Comparing to the single-variate case (Figure 4), where $p = 1$ achieves $\lambda_y \approx 1$ even at $\mathrm{SNR}_x = 1$, we see … than both t…

The results show a fascinating "Goldilocks" effect with dimensionality (p):

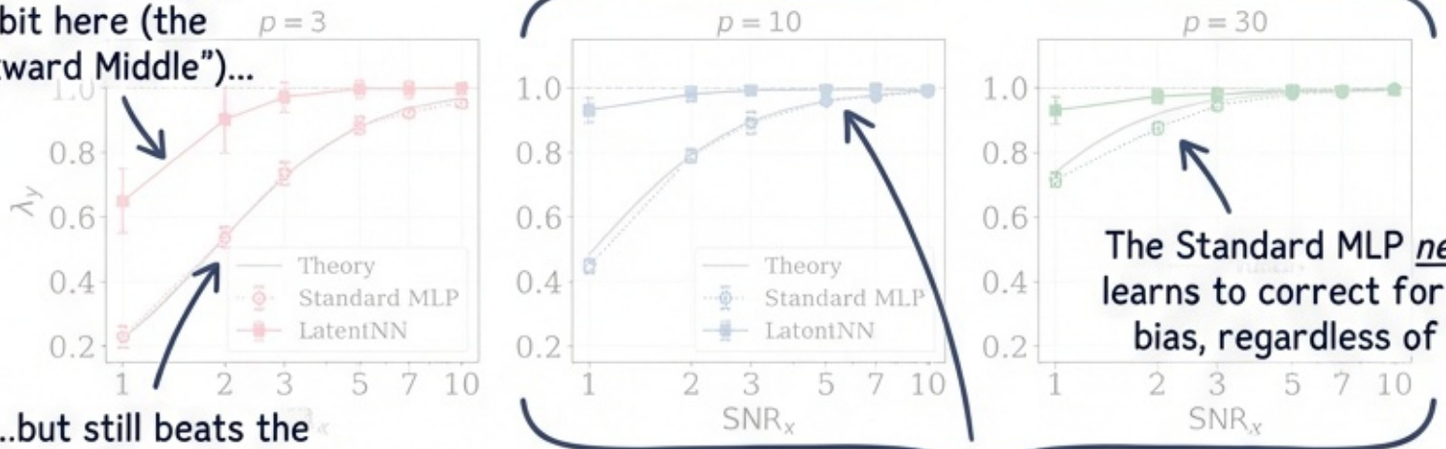


p=1 (Easy) **p=3 (Hard!)** **p=10 (Easier again!)**

At high p, correlated features "lean on each other", providing redundancy that helps constrain the latent values. It's actually easier than intermediate p!

Here's the visual proof of the "tug-of-war" from the last page!

LatentNN struggles a bit here (the "Awkward Middle")...



...but still beats the Standard MLP, which is stuck on the biased theory curve.

The Standard MLP *never* learns to correct for the bias, regardless of p.

Strength in Numbers! With more features (p=10, 30), LatentNN's correction mechanism works almost perfectly ($\lambda_y \approx 1$).

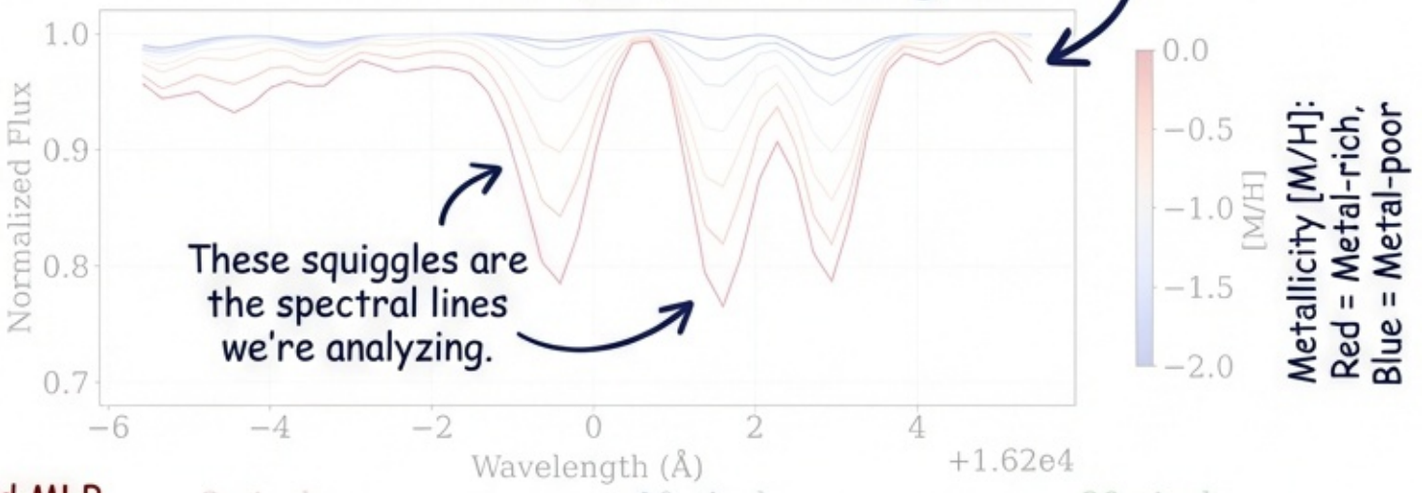
OK, enough toy models. Let's try a REAL astronomical problem!

The challenge: Real stellar spectra are noisy, and our physical models aren't perfect. This combination is a recipe for attenuation bias!

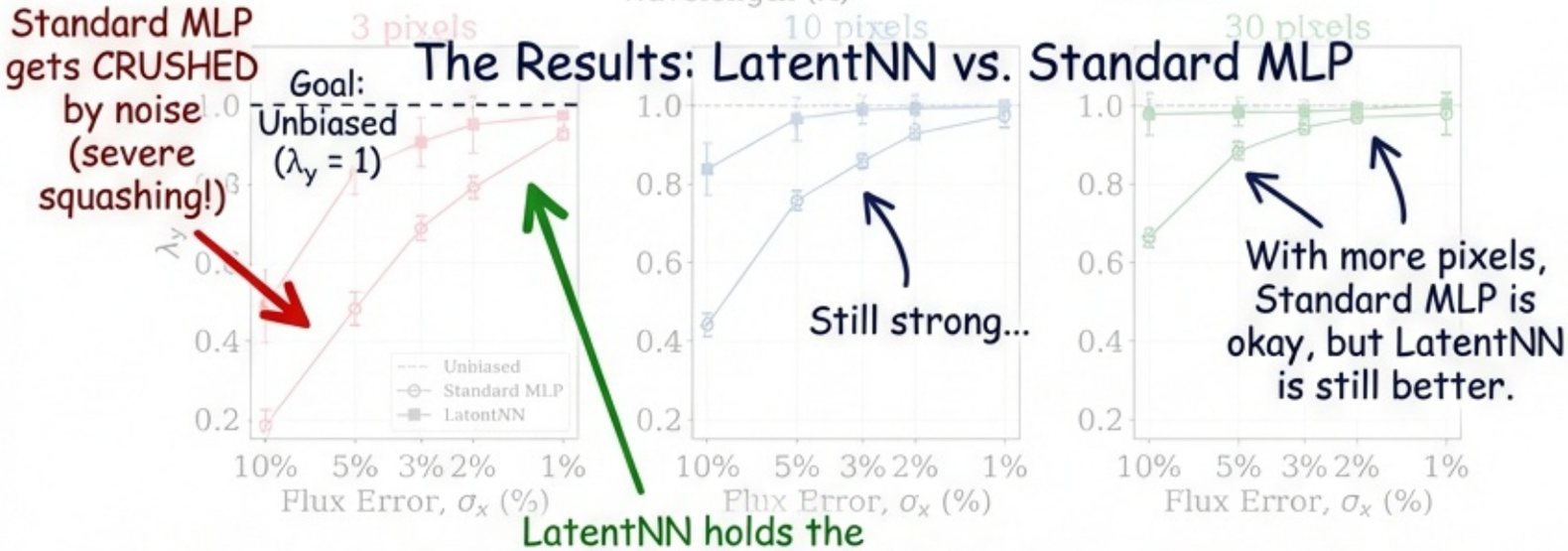
Synthetic Spectra from "The Payne" → *Add Realistic Noise* → *Test LatentNN* / *Test MLP*

We're using a sophisticated emulator to create realistic test data, focusing on tricky elements like Iron (Fe).

THE FINAL SHOWDOWN:
Real(ish) Astronomy Data!



The Results: LatentNN vs. Standard MLP



LatentNN holds the line! Near-perfect recovery ($\lambda_y \approx 1$).

This is why the '3 pixels' plot is so important! Real data is often sparse.

Adding realistic noise to challenge the models.

Conclusion: LatentNN reliably corrects attenuation bias, even with noisy, limited data typical of real surveys. It works!

Oo.

The bottom panels show λ_y versus flux error for each pixel configuration. The behavior mirrors the synthetic multivariate experiments (Section 5.2): the standard MLP exhibits attenuation that worsens with fewer pixels and higher noise, while LatentNN provides robust correction when sufficient pixels are available. For 10 and 30 pixels, LatentNN maintains $\lambda_y \gtrsim 0.95$ across all tested noise levels. For 3 pixels, however, correction degrades at high noise: $\lambda_y \approx 0.5$ at 10% flux error and $\lambda_y \approx 0.8$ at 5% flux error. This matches the intermediate-dimensionality challenge discussed in Section 5.2—with few pixels, the posterior over latent values is complex yet there is insufficient redundancy to leverage. Even so, LatentNN outperforms the standard MLP across all configurations: at 10% flux error with 3 pixels, LatentNN achieves $\lambda_y \approx 0.5$ compared to the standard MLP's $\lambda_y \approx 0.2$.

For spectral features with $\sigma_{\rm range} \approx 0.10$–$0.20$ (strong lines), flux errors of $\sigma_x \lesssim 3$–5% yield effective $\mathrm{SNR}_x \gtrsim 2$ after accounting for the factor of $\sim$2 reduction from quadratic features—consistent with our findings throughout that LatentNN works well for $\mathrm{SNR}_x \geq 2$. For weaker features with $\sigma_{\rm range} \approx 0.05$, correspondingly lower flux errors ($\sigma_x \lesssim 1$–2%) are needed. At higher noise levels with few informative pixels, careful hyperparameter tuning becomes essential, and users should expect some residual bias.

7. DISCUSSION

Implications for Spectroscopic Surveys

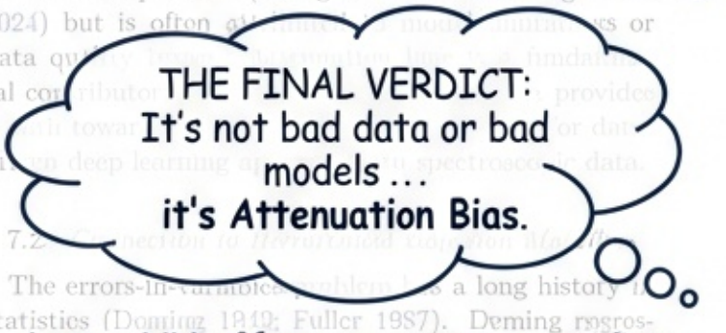
The approach of transferring stellar parameters and abundances from high-resolution surveys (Holtzman et al. 2015; Majewski et al. 2017) to low-resolution surveys like LAMOST (Cui et al. 2012; Zhao et al. 2012), DESI (DESI Collaboration et al. 2016), and Gaia XP (Andrae et al. 2023; Fallows & Sanders 2024; Khalatyan et al. 2024; Hattori 2025; Yang et al. 2025) has become standard practice—but these applications operate precisely in the regime where attenuation bias is severe.

Even at APOGEE resolution, attenuation bias is non-negligible at moderate SNR. For lower-resolution surveys where individual spectral features are blended and per-pixel $\sigma_{\rm range}$ is smaller, the ratio $\sigma_x/\sigma_{\rm range}$ is larger and attenuation bias is more severe. Crucially, $\sigma_{\rm range}$ scales approximately proportionally with spectral resolution R: a survey like LAMOST ($R \sim 1800$) has $\sigma_{\rm range}$ roughly 10 times smaller than APOGEE ($R \sim 22500$). This means that even at high conventional SNR ~ 100 ($\sigma_x \sim 1\%$), the effective SNR_x remains in the regime where attenuation bias is significant.

These huge surveys operate '*exactly*' in the high-noise regime where this bias is devastating. They are all affected.

systematically predicted to be more metal-rich, making the rarest objects harder to identify. This bias directly affects the inferred metallicity distribution function (MDF), particularly its metal-poor tail, which is crucial for constraining the formation history of the proto-Milky Way (Rix et al. 2022; Chen et al. 2024, 2025). If attenuation bias compresses the metallicity scale, the true slope of the metal-poor MDF tail would be steeper than observed, with implications for inferred gas masses, star formation efficiencies, and inflow histories in galactic chemical evolution models. Stellar ages suffer from compression at both ends, with the oldest stars underestimated and the youngest overestimated. Distances are similarly affected, leading to systematic errors in the inferred geometry and dynamics of the Milky Way. Obtaining better training labels does not solve the problem because the bias arises from noisy inputs, not labels.

When information is concentrated in sparse spectral features such as individual absorption lines, attenuation is most severe. Consider an element like K or V with only 2–3 usable lines, compared to [illegible] with dozens of strong features: K and V abundances are effectively in the low-dimensional regime where $\sigma_x/\sigma_{\rm range}$ is large, while Fe benefits from the high-dimensional correlated regime. This pattern—elements with fewer lines showing larger systematic errors at extreme values—has been observed in practice (Xiang et al. 2019; Zhang et al. 2024) but is often attributed to model limitations or data quality issues rather than fundamental statistical [illegible] provides a path toward [illegible] deep learning applications with spectroscopic data.



7.2. *Connection to Errors-in-Variables Regression*

The errors-in-variables problem has a long history in statistics (Deming 1943; Fuller 1987). Deming regres-

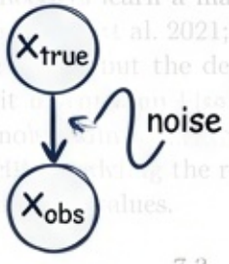
LatentNN offers a way out, providing a path to unbiased science from noisy spectral data. Case closed!

Putting it in context: How does LatentNN compare to other ideas?

– noise-augmented data

– Denoising autoencoders

These common tricks *help* with robustness but DON'T fix the fundamental BIAS. We address the root cause.



The core idea: Treat true values as "hidden" variables we need to to infer.

AHA MOMENT!
LatentNN isn't just "denoising" inputs. It's using a principled Bayesian framework to find the *most likely* true values given the noisy observations and the learned model.
The *network* parameters learn the correct relationship, even if individual latent values aren't perfect. That's the key!

The principle is flexible! Here's how it works with more complex noise (heteroscedastic).

GENERALIZATIONS

And for classification tasks! Just swap out the loss function (e.g., CrossEntropy) but keep the latent variable part.

Bottom line: LatentNN gives us a rigorous, general way to handle noisy data and get unbiased science out of our surveys. Let's put it to work!

its inputs. Convolutional neural networks for imaging data, transformers for sequential data, or any other architecture can be used by simply treating the input as learnable parameters with an appropriate latent likelihood term.

7.4. *Limitations*

The method requires specifying both input uncertainties σ_x (or $\sigma_{x,i}$ in the beteroscedastic case) and output uncertainties σ_y. In practice, σ_y may need to be estimated based on the noise model of the instrument, while σ_y reflects label uncertainties or, when labels are precise, a nominal value that sets the relative weighting in the loss function. Misspecified σ_x or σ_y leads to suboptimal correction: underestimating σ_x canses undercorrection, while overestimating it can cause overcorrection. A possible remedy is adding jitter terms to be optimized along with the other parameters.

LatentNN performs optimieation rather than sampling. We obtain point estimates of $x_{\mathrm{latent},i}$ and θ, but characterizing the full posterior over the high-dimensional space $(\theta, \mathrm{x}_{\mathrm{latcut}})$ is challenging. Full Bayesian treatment of neural networks is alrendy difficult, and adding the latent variables makes it more so. Approximate Bayesian approaches such as dropout (Hinton et al. 2012; Srivastava et al. 2014) or variational inference (Kingma & Welling 2013) may provide practical uncertainty estimates.

The increased number of parameters also leads to a more complex loss landscape. The optimization may converge to different solutions from different initializntions, particularly at low SNR where the problem is ill-conditioned. Careful hyperparameter tuning, especially of the weight decay parameter as discussed in Section 5, is often necessary.

The computational cost scales as $\mathcal{O}(N \times p)$, introducing N additional parameters per input dimenslon. For full spectra with thousands of pixels and training sets of tens of thousands of stars, this becomes a consideration. Scalability requires care, particularly for high-dimensional inference problems. Possible strategies include using subsets of informative pixels. Dimensionality reduction before LatentNN is less straightforward because the noise properties in the reduced space may not be well characterieed. Despite these limitations, the

from spectra to estimating photometric redshifts from imaging data. These applications often involve low-SNR observations where measurement uncertainties are non-negligible. When inputs are noisy, attenuation bias becomes a fundamental concern.

This

for from the

cient slopes biased toward zero by a factor that scales with $\sigma_x/\sigma_{\mathrm{range}}$. Increasing model complexity does not

applications involve high-SNR inputs where the effect is negligible. Astronomy is different, and as neural net-

ence, clasaical

in-variables become important.

exput

the

ral-

asly

learns the correct input-output relationship rather than one diluted by measurement noise.

We propose LatentNN as a practical implementation

optimizes these latent values into the network parame-

ing regression for the same data, on multivariate inputs with varying correlation structure, and on realistic spec-

attenuation bias, particularly for $\mathrm{SNR}_x \gtrsim 2$, with the

prov ce with neural networks in the low-SNR regime that characterizes much of astronomical data. The method is relevant for spectroscopic surveys, photometric applications, and any data-driven model where inputs carry non-negligible measurement uncertainty.

Crucial requirement: We need good estimates of the noise (σ_x, σ_y)! Garbage in, garbage out applies to noise estimates too.

It's computationally expensive! Scales with N×p. Big data = big compute load.

Optimization is trickier with so many parameters to juggle.

HEADS UP! It's not magic:
- Needs accurate noise estimates
- Harder to train
- Computationally heavy

The problem: Neural nets inherit the 'squashing' bias from classic regression when inputs are noisy.

Our solution: LatentNN! It learns the 'true' inputs as hidden variables, fixing the bias at the source.

The result: It works! We get unbiased, accurate models even with noisy data.

The big picture: This is a game-changer for astronomy, where noisy data is the norm.

WRAPPING IT UP:

Attenuation bias is a real danger in ML with noisy data, leading to wrong science. LatentNN is a rigorous, principled way to fix it by wedding modern deep learning with classic statistical ideas. It's not free, but for precision science, it's worth it!

Now go fix your models! – The Authors

The final word: Don't let noisy data fool your fancy neural net! Accounting for measurement errors is key to getting "real" science out.

Neural network modeling is a powerful tool, but it is not a panacea. Neural networks inherit the same limitations as classical statistical methods when those limitations arise from the data rather than the model. Attenuation bias is one such limitation. Recognizing when classical statistical insights apply to modern machine learning methods, and adapting solutions accordingly, is essential for reliable scientific inference in astronomical research.

CODE AVAILABILITY

Grab the code and start fixing your biases! →

We provide a reference implementation of LatentNN at https://github.com/tingyuansen/LatentNN. The code demonstrates the method for simple fully-connected architectures as presented in this paper. The approach is general and can be adapted to any differentiable architecture by treating the input as learnable parameters with an appropriate latent likelihood term.

ACKNOWLEDGMENTS

I thank David Weinberg for the encouragement that led to Paper I of this series. The original idea about errors-in-variables stemmed from my teaching of Astron S550 (Advanced Data Analysis), during which I decided to discuss attenuation bias as explored in Paper I. While preparing for that lecture, I was inspired to read more into the errors-in-variables literature, also suggested by an anonymons referee during Paper I. Further discussion with Jeffrey Newman and Brett Andrews during a colloqnium at UPitt Inspired the extension of this work to deep learning, which ultimately led to this paper. I also thank Hans-Walter Rix for useful discussion.

This research was supported by NSF Grant AST-2406729 and a Humboldt Research Award from the Alexander von Humboldt Foundation.

Claude 4.5 was used for assistance with code development and manuscript copy-editing. The scientific content, analysis, and conclusions are the author's, and all LLM-assisted changes were verified by the author.

Science is a team sport! Big thanks to everyone who bounced ideas around (probably over too much coffee).

A nod to our AI assistants too 🤖

APPENDIX

APPENDIX: For the math folks, here's the derivation of the 'squashing'.

A. DERIVATION OF ATTENUATION BIAS

Consider a true linear relationship $y_{\rm true} = \beta x_{\rm true}$. Our observations are

$$x_{\rm obs} = x_{\rm true} + \delta_x, \tag{A1}$$

$$y_{\rm obs} = \beta x_{\rm true} + \delta_y, \tag{A2}$$

where δ_x and δ_y are independent measurement errors with variances σ_x^2 and σ_0^2.

The standard least squares estimator is $\hat{\beta} = {\rm Cov}(x_{\rm obs}, y_{\rm obs})/{\rm Var}(x_{\rm obs})$. Expandit

$$\begin{aligned} {\rm Cov}(x_{\rm obs}, y_{\rm obs}) &= {\rm Cov}(x_{\rm true} + \delta_x, \beta x_{\rm true} + \delta_y) \qquad \text{(A3)} \\ &= \beta\, {\rm Var}(x_{\rm true}) = \beta \sigma^2_{\rm range}, \qquad \text{(A4)} \end{aligned}$$

See? The math proves it. Noise in 'x' makes the estimated slope ($\hat{\beta}$) smaller than the true slope (β). That's attenuation bias in a nutshell.

since measurement errors are independent of true values and of each other. For the denominator:

$${\rm Var}(x_{\rm obs}) = {\rm Var}(x_{\rm true}) + {\rm Var}(\delta_x) = \sigma^2_{\rm range} + \sigma^2_x. \tag{A5}$$

The expected value of the standard estimator is therefore

$$\mathbb{E}[\hat{\beta}] = \beta \frac{\sigma^2_{\rm range}}{\sigma^2_{\rm range} + \sigma^2_x} = \beta \cdot \lambda_\beta, \tag{A6}$$

where $\lambda_\beta = 1/(1 + (\sigma_x/\sigma_{\rm range})^2) < 1$ is the attenuation factor. The estimated slope is systematically biased toward zero.

B. DERIVATION OF DEMING REGRESSION

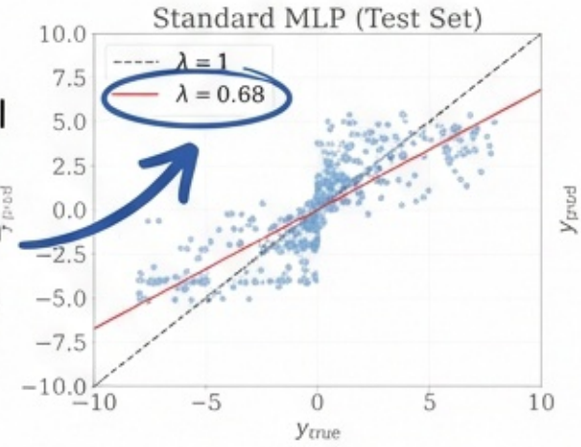
And here's the classic statistical fix that inspired it all: Deming Regression.

To correct attenuation bias, we treat the true values $\{x_{{\rm true},i}\}$ as latent variables to be estimated alongside β. The joint likelihood, conditioned on the true values, is

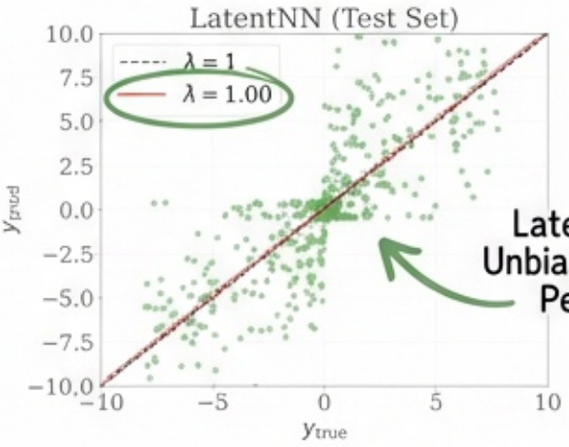
$$\mathcal{L} = \prod_{i=1}^{N} \mathcal{N}(x_{{\rm obs},i} | x_{{\rm true},i}, \sigma_x^2) \cdot \mathcal{N}(y_{{\rm obs},i} | \beta x_{{\rm true},i}, \sigma_y^2). \tag{B7}$$

The key idea: Treat the 'true' values ($x_{{\rm true},i}$) as unknowns (latent variables) and solve for them together with the slope.

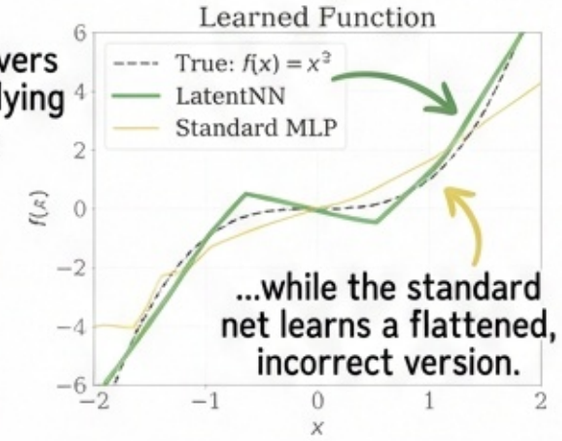
Seeing is believing: The non-linear challenge!



Standard Neural Net is STILL squashed by noise, even with a curvy function! Look at that bias! λ < 1.

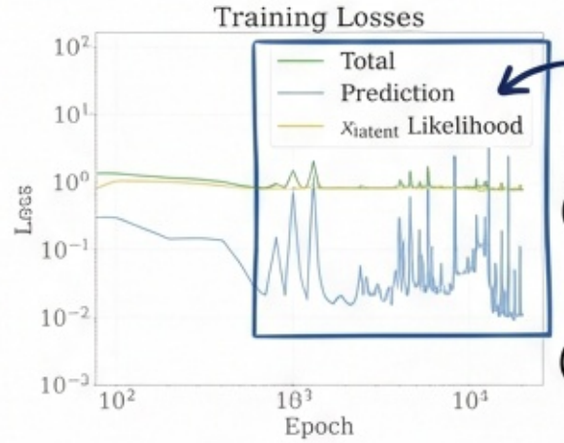


LatentNN nails it! Unbiased predictions. Perfect slope! λ = 1.



LatentNN discovers the TRUE underlying curve (x^3)...

...while the standard net learns a flattened, incorrect version.



The training "tug-of-war" in action! It's balancing fitting the data (Prediction loss) vs. finding plausible latent values (Likelihood loss).

Figure 7. LatentNN applied to a nonlinear function $f(x) = x^3$ at $\mathrm{SNR}_y = 2$. Top: Predicted versus true y on the test set for standard MLP (left, $\lambda = 0.68$) and LatentNN (right, $\lambda = 1$). Bottom left: Learned function compared to the true cubic. Bottom right: Training losses for LatentNN, showing the trade-off between prediction accuracy and latent variable likelihood.

Back to the MATH ZONE:

How do we find those hidden "true" values?

This is the "cost function" we want to minimize. It penalizes estimates that are far from the observed noisy data (both x and y).

Expanding the normal distributions and taking the n

$$E(\{x_{\mathrm{true},i}\}, \beta) = \sum_{i=1}^{N} \left[\frac{(x_{\mathrm{obs},i} - x_{\mathrm{true},i})^2}{2\sigma_x^2} + \frac{(y_{\mathrm{obs},i} - \beta x_{\mathrm{true},i})^2}{2\sigma_y^2} \right]. \quad \text{(B6)}$$

For a fixed β, each $x_{\mathrm{true},i}$ appears only in the ith term, so we can optimize them independently. Taking the derivative and setting it to zero:

$$\frac{\partial E}{\partial x_{\mathrm{true},i}} = -\frac{x_{\mathrm{obs},i} - x_{\mathrm{true},i}}{\sigma_x^2} - \frac{\beta(y_{\mathrm{obs},i} - \beta x_{\mathrm{true},i})}{\sigma_y^2} = 0.$$

Rearranging and letting $r = \sigma_y^2/\sigma_x^2$ denote the error variance ratio:

$$x_{\mathrm{true},i} = \frac{r\, x_{\mathrm{obs},i} + \beta y_{\mathrm{obs},i}}{r + \beta^2}. \quad \text{(B10)}$$

This can be rewritten as a weighted average:

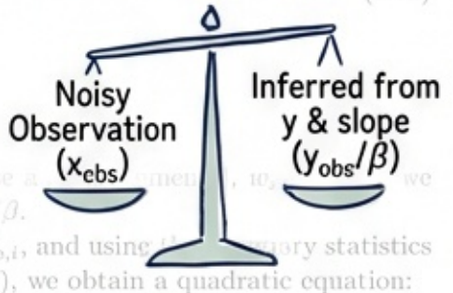
$$x_{\mathrm{true},i} = w_x \cdot x_{\mathrm{obs},i} + w_y \cdot \frac{y_{\mathrm{obs},i}}{\beta},$$

where $w_x = r/(r+\beta^2)$ sum to unity. When $r \gg 1$ ements), $w_y \approx 1$ and we trust y_{obs}/β.

Weight depends on x-noise...

...vs y-noise. Clever!

THE KEY INSIGHT!
The "true" x value is just a **smart weighted average.**



Taking the derivative of E with respect to β, substituting the exp $x_{\mathrm{true},i}$, and using the sample statistics $S_{xx} = \sum_i (x_{\mathrm{obs},i} - \bar{x})^2$, $S_{yy} = \sum_i (y_{\mathrm{obs},i} - \bar{y})^2$, and $S_{xy} = \sum_i (x_{\mathrm{obs},i} - \bar{x})(y_{\mathrm{obs},i} - \bar{y})$, we obtain a quadratic equation:

$$S_{xy}\beta^2 - (S_{yy} - rS_{xx})\beta - rS_{xy} = 0. \quad \text{(B12)}$$

Finally, we plug those "true" x's back in to solve for the slope β (it gets a bit messy...).

Here's that messy result! The "true" slope β, finally revealed.

Applying the quadratic formula and taking the positive root:

Calculating...

$$\hat{\beta} = \frac{S_{yy} - rS_{xx} + \sqrt{(S_{yy} - rS_{xx})^2 + 4rS_{xy}^2}}{2S_{xy}}. \tag{B13}$$

In the limit of negligible measurement error (formally $r \to \infty$), this reduces to $\hat{\beta} \to S_{xy}/S_{xx}$, the ordinary least squares estimator.

C. EXTENSION TO NONLINEAR FUNCTIONS

This text explains the cool plots on the LAST PAGE!

The main text focuses on linear functions, where the attenuation factor $\lambda_y = 1/(1 + (\sigma_s/\sigma_{\text{range}})^2)$. However, the errors-in-variables problem and the LatentNN correction are more general and apply to arbitrary nonlinear functions.

Testing on a curvy function, not just a straight line.

To demonstrate this, we test LatentNN on a cubic polynomial $f(x) = x^3$. [illegible] $\times$ 256 hidden nnits) to ensure the network can represent the nonlinear function accur [illegible] $\text{SNR}_y = 2$, a challenging regime where noise equals 50% of the signal range. The same weight decay hyperparameter search procedure from Section 5 is applied.

The standard MLP exhibits severe attennation, with $\lambda_y \approx 0.65$—oven worse than the linear case at the same SNR, as derived in Paper 1. This occurs because measurement errors in x are amplified by the nunlinear function: an error δ_s produces a prediction error of approximately $f'(x)\delta_s$, which varies across the inpot domain. Near the edges of the training range where $|f'(x)) = 3x^2$ is largest, the effective noise is amplified.

LatentNN corrects this bias. The bottom-left panel shows the learned functions: the standard MLP learns a flattened version of the cobic, while LatentNN recovers the true function shape. This demonstrates that the latent variable framework extends naturally beyond linear regression to nonlinear function approximation.

LatentNN fixes it! It finds the true underlying curve. This nethod's just for simple the linear problems. It works for complex, real-world non-linear relationships too!

Andrae, R., Rix, H.-W., & Chandra, V. 2023a, ApJS, 267, 8, doi: 10.3847/1538-4365/acd53e

Andrae, R., Fouesneau, M., Sordo, R., et al. 2023b, A&A, 674, A27, doi: 10.1051/0004-6361/202243462

Bishop, C. M. 2006, Pattern Recognition and Machine Learning (Information Science and Statistics) (Berlin, Heidelberg: Springer-Verlag)

Blanco-Cuaresma, S., Soubiran, C., Heiter, U., & Jofré, P. 2014, A&A, 569, A111, doi: 10.1051/0004-6361/201423945

Brown, T. B., Mann, B., Ryder, N., et al. 2020, arXiv e-prints, arXiv:2005.14165, doi: 10.48550/arXiv.2005.14165

Carroll, R. J., Ruppert, D., & Stefanski, L. A. 1995, Measurement Error in Non-linear Models (New York: Chapman and Hall)

Carroll, R. J., Ruppert, D., Stefanski, L. A., & Crainiceanu, C. M. 2006, Measurement error in nonlinear models: A modern perspective, Second edition (Chapman and Hall)

Chen, B., Orkney, M. D. A., Ting, Y.-S., & Hayden, M. R. 2025, arXiv e-prints, arXiv:2511.18901, doi: 10.48550/arXiv.2511.18901

Chen, B., Ting, Y.-S., & Hayden, M. 2024, PASA, 41, e063, doi: 10.1017/pasa.2024.56

Collister, A. A., & Lahav, O. 2004, PASP, 116, 345, doi: 10.1086/383254

Cranmer, K., Brehmer, J., & Louppe, G. 2020, Proceedings of the National Academy of Science, 117, 30055, doi: 10.1073/pnas.1912789117

Cybenko, G. 1989, Mathematics of Control, Signals, and Systems, 2, 303, doi: 10.1007/BF02551274

Deming, W. E. 1943, Statistical adjustment of data (Wiley)

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. 2018, arXiv e-prints, arXiv:1810.04805, doi: 10.48550/arXiv.1810.04805

Fabbro, S., Venn, K. A., O'Briain, T., et al. 2018, MNRAS, 475, 2978, doi: 10.1093/mnras/stx3298

Fallows, C. P., & Sanders, J. L. 2024, MNRAS, 531, 2126, doi: 10.1093/mnras/stae1303

Frost, C., & Thompson, S. G. 2002, Journal of the Royal Statistical Society Series A: Statistics in Society, 163, 173, doi: 10.1111/1467-985X.00164

Fuller, W. A. 1987, Measurement Error Models (New York: Wiley)

García Pérez, A. E., Allende Prieto, C., Holtzman, J. A., et al. 2016, AJ, 151, 144, doi: 10.3847/0004-6256/151/6/144

Gelman, A., Carlin, J. B., Stern, H. S., et al. 2013, Bayesian data analysis (3rd ed.) (Chapman and Hall)

The giants whose shoulders we stand on.
(Also, a great reading list for later.)

And the list goes on... Seriously, science is a massive collaborative effort over decades!

Gheller, C., & Vazza, F. 2022, MNRAS, 509, 990, doi: 10.1093/mnras/stab3044

Golub, G. H., & Van Loan, C. F. 1980, SIAM Journal on Numerical Analysis, 17, 883

Guo, J., Bai, X., Deng, Y., et al. 2020, SoPh, 295, 3, doi: 10.1007/s11207-019-1573-9

Hattori, K. 2025, ApJ, 980, 90, doi: 10.3847/1538-4357/ad9656

He, K., Zhang, X., Ren, S., & Sun, J. 2016, in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1, doi: 10.1109/CVPR.2016.90

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. 2012, arXiv e-prints, arXiv:1207.0580, doi: 10.48550/arXiv.1207.0580

Ho, A. Y. Q., Ness, M. K., Hogg, D. W., et al. 2017, ApJ, 836, 5, doi: 10.3847/1538-4357/836/1/5

Hoerl, A. E., & Kennard, R. W. 1970, Technometrics, 12, 55

Holtzman, J. A., Shetrone, M., Johnson, J. A., et al. 2015, AJ, 150, 148, doi: 10.1088/0004-6256/150/5/148

Hon, M., Huber, D., Kuszlewicz, J. S., et al. 2021, ApJ, 919, 131, doi: 10.3847/1538-4357/ac14b1

Hornik, K., Stinchcombe, M., & White, H. 1989, Neural Networks, 2, 359

Hoyle, B. 2016, Astronomy and Computing, 16, 34, doi: 10.1016/j.ascom.2016.03.006

Huertas-Company, M., & Lanusse, F. 2023, [illegible] e001, doi: 10.1017/pasa.2022.55

Kelly, B. C. 2007, ApJ, 665, 1489, doi: 10.[illegible]

Khalatyan, A., Anders, F., Chiappini, C., [illegible] 691, A98, doi: 10.1051/0004-6361/2024[illegible]

Kingma, D. P., & Welling, M. 2013, arXiv e-prints, arXiv:1312.6114, doi: 10.48550/arXiv.1[illegible]

Kurucz, R. L. 1970, SAO Special Report[illegible]

—. 1993, SYNTHE spectrum synthesis programs and line data

—. 2005, Memorie della Societa Astronomica Italiana Supplementi, 8, 14

—. 2017, ATLAS9: Model atmosphere program with opacity distribution functions, Astrophysics Source Code Library. http://ascl.net/1710.017

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. 1998, Proceedings of the IEEE, 86, 2278

Leung, H. W., & Bovy, J. 2019, MNRAS, 483, 3255, doi: 10.1093/mnras/sty3217

Li, J., Wong, K. W. K., Hogg, D. W., Rix, H.-W., & Chandra, V. 2024, ApJS, 272, 2, doi: 10.3847/1538-4365/ad2b4d

Li, R., Napolitano, N. R., Feng, H., et al. 2022a, A&A, 666, A85, doi: 10.1[illegible]

Li, X., Wang, Z., Zeng, S., et al. 2022b, Research in Astronomy and Astrophysics, 22, 065018, doi: 10.1088/1674-4527/ac65e7

Lin, Q., Fouchez, D., Pasquet, J., et al. 2022, A&A, 662, A36, doi: 10.1051/0004-6361/202142751

Loredo, T. J. 2004, in American Institute of Physics Conference Series, Vol. 735, Bayesian Inference and Maximum Entropy Methods in Science and Engineering: 24th International Workshop on Bayesian Inference and Maximum Entropy Methods in Science and Engineering, ed. R. Fischer, R. Preuss, & U. V. Toussaint (AIP), 195–206, doi: 10.1063/1.1835214

Luo, A. L., Zhao, Y.-H., Zhao, G., et al. 2015, Research in Astronomy and Astrophysics, 15, 1095, doi: 10.1088/1674-4527/15/8/002

Majewski, S. R., Schiavon, R. P., Frinchaboy, P. M., et al. 2017, AJ, 154, 94, doi: 10.3847/1538-3881/aa784d

Ness, M., Hogg, D. W., Rix, H.-W., Ho, A. Y. Q., & Zasowski, G. 2015, ApJ, 808, 16, doi: 10.1088/[illegible]/808/1/16

Odewahn[illegible] Pennington, R. L., Hun[illegible] V. A. 1992, AJ, 103, 318, doi:[illegible]

Op[illegible]al. 2023, arXiv e-prints, arXiv:2[illegible]80/arXiv.2303.08774

Pdl, B., & Dobes, L. 2034, Astronomische Nachrichten, 345, e20240049, doi: 10.1002/asna.20240049

[illegible]uet, J., Pasquet, J., Chaumont, M., & Fouchez, D. [illegible]19, A&A, 627, A21, doi: 10.1051/0004-6361/201834473

[illegible]. H.-W., Ting, Y.-S., Conroy, C., & Hogg, D. W. 2016, [illegible]L, 826, L25, doi: 10.3847/2041-8205/826/2/L25

[illegible]H.-W., Chandra, V., Andrae, R., et al. 2022, ApJ, 941, [illegible]doi: 10.3847/1538-4357/ac9e01

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. 1986, Nature, 323, 533

Simonyan, K., & Zisserman, A. 2014, arXiv e-prints, arXiv:1409.1556, doi: 10.48550/arXiv.1409.1556

Spearman, C. 1904, The American Journal of Psychology, 15, 72, doi: 10.2307/1412159

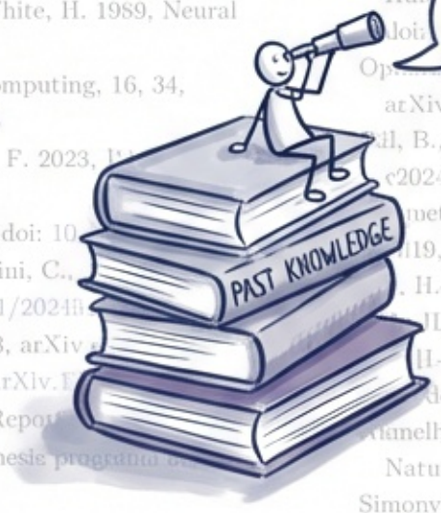
Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. 2014, Journal of Machine Learning Research, 15, 1929

Storrie-Lombardi, M. C., Lahav, O., Sodre, Jr., L., & Storrie-Lombardi, L. J. 1992, MNRAS, 259, 8P, doi: 10.1093/mnras/259.1.8P

Straumit, I., Tkachenko, A., Gebruers, S., et al. 2022, AJ, 163, 236, doi: 10.3847/1538-3881/ac5f49

Szegedy, C., Liu, W., Jia, Y., et al. 2014, arXiv e-prints, [illegible]

[illegible]Doklady, 4, 1035



Thanks for sticking with me to the end!
Now, go forth and stand on some shoulders of your own.
(And maybe read some of these papers!)

A final few shout-outs to the community! Science is a team sport, after all.

Wang, E. X., Nordlander, T., Asplund, M., et al. 2021, MNRAS, 500, 2159, doi: 10.1093/mnras/staa3351
Winecki, D., & Kochanek, C. S. 2024, ApJ, 971, 61, doi: 10.3847/1538-4357/ad5a0b
Wong, K. W. K., Ng, K. K. Y., & Berti, E. 2020, arXiv e-prints, arXiv:2007.10350, doi: 10.48550/arXiv.2007.10350
Xiang, M., Ting, Y.-S., Rix, H.-W., et al. 2019, ApJS, 245, 34, doi: 10.3847/1538-4365/ab3364
Xiang, M., Rix, H.-W., Ting, Y.-S., et al. 2022, A&A, 662, A66, doi: 10.1051/0004-6361/202141570
Yang, L., Yuan, H., Huang, B., et al. 2025, ApJS, 279, 7, doi: 10.3847/1538-4365/add5e3
Zhang, M., Xiang, M., Ting, Y.-S., et al. 2024, ApJS, 273, 19, doi: 10.3847/1538-4365/ad51dd
Zhang, X., Green, G. M., & Rix, H.-W. 2023, MNRAS, 524, 1855, doi: 10.1093/mnras/stad1941

Carroll, R. J., Ruppert, D., & Stefanski, L. A. 1988, Measurement Error in Non-linear Models (New York: Chapman and Hall)
Carroll, R. J., Ruppert, D., Stefanski, L. A., & Crainiceanu, C. M. 2006, Measurement error in nonlinear models: A modern perspective, Second edition (Chapman and Hall)
Chen, B., Orkney, M. D. A., Ting, Y.-S., & Hayden, M. R. 2025, arXiv e-prints, arXiv:2511.18901, doi: 10.48550/arXiv.2511.18901
Chen, B., Ting, Y.-S., & Hayden, M. 2024, PASA, 41, e063, doi: 10.1017/pasa.2024.55

PS: Don't forget to check out the code repository mentioned earlier!