Epistatic-degree-regularized neural networks for sequence-based prediction

Renzo G. Soatto, Clara Fannjiang, and Jennifer Listgarten

University of California, Berkeley; Berkeley CA 94720, USA
{renzo,clarafy,jennl}@berkeley.edu

Abstract. Biological sequence-based prediction, such as protein property prediction from sequence, has progressed over the years from the use of relatively simple linear models to complex neural networks. However, the scale of supervised training data has not matched that of other fields, motivating the need for strong biological sequence-specific inductive biases to enable strong performance by the neural networks. For example, recent work has shown that regularizing a neural network in an "epistatic" basis—the Walsh-Hadamard basis—can yield improved predictive performance for a neural network as compared to more standard regularization such as on the L_2-norm of neural network weights. However, past instantiations of WH inductive biases do not account for the knowledge that higher-order epistatic terms are less likely than lower order ones. Additionally, these WH-based methods can only be applied natively to binary sequence data and have poor scaling properties. To address these two shortcomings, we introduce a new kernel-based regularization of neural networks. Our method leverages a variant of the ANOVA kernel that we develop for categorical data, which can be computed in linear time with the sequence length. Through simulations and analysis of real experimental data, we demonstrate the advantages provided by our method in comparison to other prediction approaches.

Keywords: supervised prediction on sequences · inductive bias · Fourier regularization · kernel · neural networks · Gaussian processes regression · implicit layer.

1 Supervised learning on sequences

Supervised learning on the domain of amino acid and nucleotide sequences is pervasive in many problems in biology. For example, predicting protein properties such as stability, binding, or catalytic activity from protein sequence is a key element of machine learning-guided protein engineering. Other examples include predicting the expression level of a gene from its sequence or the pathogenicity of a virus from its sequence. Although predictive tasks may also make use of structure, most predictive tasks will always require sequence, the topic of our work herein. Over the years, these classification and regression tasks have employed a wide range of models, from linear models with few or no epistastic terms to decision trees as well as neural networks, especially recently, as data and compute volume have increase dramatically. As in many other fields, neural networks have proven themselves competitive for such tasks [9]; we will focus on these models. Architectural innovations and inductive biases in neural networks have been largely tailored to problems in computer vision and Natural Language Processing, with one notable exception being the incorporation of physical constraints/biases, typically as they pertain to rotational symmetries [8, 12], such as in structure prediction [13]. To make faster progress, a thoughtful revisiting of appropriate inductive biases for the domain at hand may prove useful.

For neural networks that operate on discrete biological tokens such as amino acids or nucleotides, it has been shown theoretically under some conditions that neural networks on discrete inputs tend to learn lower degree epistastic terms earlier in the learning process[29, 10]. However, there has been no explicit work to formally regularize a neural network with the knowledge that most biological functions depend mostly on lower order terms, such as non-epistatic terms, followed next by pairwise epistasis, and so forth—that is, we expect on average that higher-order epistastic terms are less likely than lower-order ones. Rather, application of neural networks in biology almost uniquely regularizes the neural network parameters themselves, which do not in general correspond directly to epistastic terms in a linear model where our prior knowledge lies. In other words, unlike in, for example, computer vision, where convolution captures prior knowledge about the spatial domain of images, in biological sequence learning, there is a mismatch between the "currency" of our prior knowledge and that of the parameters of the neural network. We seek to find a generally useful inductive bias for sequence-based prediction, similarly to how convolutions have been in many other domains.

Recent work has tackled the issue of imposing an inductive bias on epistatic terms for networks on biological alphabets by regularizing neural networks in the Walsh-Hadamard (WH) basis [2, 10]. However these approaches can only natively handle binary alphabets and thus do not readily extend to biological sequences comprising either four, or 20-letter alphabets. This binary limitation presents either a fundamental representation issue, or a scaling issue, if not both. More importantly, the only inductive bias these approaches can impose is that of K-sparsity on epistatic terms of *all degrees*, up to and including the degree of the length of the sequence itself. That is, the inductive bias encoded therein does not capture the knowledge that a 10^{th} degree epistatic term is generally less likely than a single-site or pairwise term, which goes against our understanding of the nature of most biological landscapes.

Our contributions herein are to mitigate these issues and others by i) leveraging a generalization of the WH basis that naturally extends to arbitrary-sized alphabets, ii) allowing the inductive bias to prefer low-degree over high-degree epistatic terms (or any weighting of the different degrees desired), and iii) bypassing explicit computation of all epistatic terms by using the kernel trick through Gaussian process regression with ANOVA-based kernels modified to handle discrete sequences. We also develop a new way to compute this discrete ANOVA kernel that is vastly more efficient, by both reducing the time complexity to linear in sequence length and optimizing it for parallel computation on GPUs. To enable our overall approach, we make use of the implicit function theorem in an "implicit layer" [3] containing the Gaussian process regression for the regularization, so that we can more efficiently perform gradient descent through the regularization term in the loss function. We call our approach *Epistatic-degree-regularized* (EDR) neural networks, or EDR-NNs for short.

2 Related Work

An overall discussion on prior work of "simplicity" biases in neural networks—primarily focused on continuous inputs and single-dimension input and outputs—can be found in Gorji *et al.* [10]. To the best of our knowledge, the first to consider "epistatic" regularizaton of neural networks was performed by leveraging efficient algorithms for K-sparsity from compressed sensing, employing an L_1-regularization on the implied epistatic terms of the neural network [2]. As mentioned earlier, this approach is not generalizable beyond binary input tokens and cannot impose different regularization on different degree epistatic terms. The presented method requires a number of approximations to be made in computing the required WH transform because of the combinatorial nature of the problem. In fact, we speculate that any method for performing this general class of regularization will similarly require some level of approximation, and the interesting and impactful questions are whether certain approximations are more useful than others, due to either accuracy or computational efficiency.

Following up on the work of Aghazadeh *et al.*, Gorji *et al.* [10] aim to perform the same regularization, but, by using a more efficient method of approximation to the WH transform used for regularization, they achieve dramatically faster runtimes than Aghazadeh *et al.*, which allows them to make fewer accuracy sacrifices in order to manage runtime. An elegant and practically useful aspect of their solution is the ability to set a hyper-parameter to gracefully trade off between the exact regularization, which is computationally impractical, to one that approximates the exactness more and more. The key idea is to use a hash function to collapse WH coefficients into 2^b bins, where this number is less than the total number of coefficients, 2^n, for pseudo-boolean functions on n binary features. With the hashing, their method scales with 2^b, rather than with 2^n. The extent to which important coefficients with different signs get hashed into the same bin, is the extent to which this approximation will break down.

Unlike these previous works [2, 10], both of which can impose only one specific inductive bias—K-sparsity on the WH coefficients—our proposed method is able to impose any inductive bias that can be described by the degree of the WH coefficients. This means we can, for example, impose an inductive bias such that higher-order epistatic coefficients are more likely to have smaller values, by way of a degree-specific L_2-regularization on each set of coefficients specified by degree. Furthermore, we do so without explicit computation of individual epistatic terms, instead appealing to kernels so as to estimate the overall effect of each degree of epistasis.

3 Epistatic-degree-kernel-regularized neural networks

Before delving into the details of our method, we present some background knowledge that will be useful for our method development. For clarity and accessibility of presentation, most of the technical narrative is presented for binary sequence tokens. However, we later generalize to sequences with larger alphabets. Further technical details are available in the Appendix.

3.1 Linear additive models as universal function approximators

First, note that any non-linear function $y = f(x)$, on a sequence of binary tokens, x, for $y \in \mathbb{R}$, can be written as a linear additive model that contains all possible epistatic terms. That is, for a sequence of length L, any such function can be represented uniquely as a linear combination of 2^L epistatic terms:

$$f(x) = \sum_{S \subseteq [L]} \alpha_S \prod_{i \in S} x_i, \tag{1}$$

where $[L] := \{1, \ldots, L\}$ and each monomial, $\prod_{i \in S} x_i$ is one interaction term of epistatic degree $|S|$. If we restrict S to contain only single positional terms ($|S| = 1$), then the model would be a "site-specific" model. If we also allow $|S| = 2$, then the model would include pairwise epistastic terms. To

enable this linear additive model to be a universal function approximator, we must allow all S, which thus includes those up to and including $|S| = L$. A more common example of a similar phenomenon is that for continuous inputs, a universal function approximator can be constructed in the form of a linear additive model with a complete polynomial basis function expansion of the features. In both cases, although this parametric form can encode any function, rarely does such a functional form yield the most accurate predictions after being trained on a finite data set, owing to the lack of inductive bias. One might consider regularizing a linear additive model in this combinatorial feature space but such models quickly become intractable as sequences grow. Instead, we seek to leverage the power of neural networks, which implicitly can cover this space, but adaptively without explicit enumeration of all monomials. Typical neural network inductive biases arise from parameter initialization, the use of gradient descent, architecture choices, L_1- and L_2-regularized weights, and dropout, for example. None of these techniques, however, enables us to regularize in the space of monomials described by Equation 1—the space where our prior biological knowledge lies.

Given a function $y = f(x)$, if we knew all values of y for all possible values of x, then computing what is known as the WH transform would yield the monomial coefficients, α_S, in Equation 1. This transform provides the foundation upon which other recent work has regularized neural networks on biological sequences [2, 10]. However, herein, we bypass such an explicit transformation, instead relying on estimating the overall effect of all epistatic terms of each degree—that is for degrees $d = 1, 2, \ldots, L$, where recall L is the sequence length, by use of the kernel trick, which only implicitly makes use of the full combinatorial set of epistatic terms. Specifically, we use a generalization of what is called an ANOVA kernel [22]. We will use the kernel trick by way of Gaussian Process regression [19], as an implicit layer [11] in the regularization loss of our neural network. This regularization loss will encode our prior epistatic beliefs at the degree level.

3.2 Desired inductive bias by way of the ANOVA kernel

The goal of the ANOVA kernel function [21] for degree, d, is to efficiently compute the inner product distance between two sequences x_1 and x_2, when they have been expanded into the full epistatic basis of Equation 1. For example, the ANOVA kernel for $d = 1$ would simply compute the inner product of the raw sequences, whereas the ANOVA kernel for $d = 2$ would compute the inner product of sequences when encoding them using *only* all pairwise features. Notably, the ANOVA kernel of degree d, only (implicitly) makes use of features of that degree, and not lower or higher, unlike the more widely-known polynomial kernels, which include all features up to and including the degree of the kernel. As a consequence, we can first use the ANOVA kernel to group together the epistatic terms by degree and then estimate the norm of the epistatic (WH) coefficients for each degree. Ultimately, this will let us regularize our neural network differently for each epistatic degree.

More formally, the ANOVA kernel function of order d, $K_d(x, x') = \langle \Phi_d(x), \Phi_d(x') \rangle$, is the inner product between the d^{th} degree epistatic expansion of sequence x and that of sequence x'. The feature map, $\Phi_d(x)$, corresponds to all $\binom{L}{d}$ epistatic terms of order d:

$$\Phi_d(x) = \left(\prod_{i \in S} x_i \right)_{S \subseteq [L], |S| = d} . \tag{2}$$

By way of dynamic programming, the ANOVA kernel can be computed with time complexity $O(Ld)$ [21], rather than the naive computation of $O\binom{L}{d}$. However, as shown in AA.2, by restricting ourselves to categorical sequences, we develop a more efficient algorithm yet, with time complexity $O(L + d)$, plus some precomputations.

We can now re-write Equation 1 with the implicit feature space of the ANOVA kernel,

$$f(x) = \sum_{d=0}^{L} \alpha_d^T \Phi_d(x), \text{where } \alpha_d \in \mathbb{R}^{\binom{L}{d}} \equiv (\alpha_S)_{S \subseteq [L], |S| = d} . \tag{3}$$

So long as the prediction algorithm accesses inputs only through their inner products in these implicit spaces, we can make use of the kernel trick. We will do so by way of the machinery of Gaussian process regression to help us regularize our neural network, which we explain in Section 3.3.

Previous work imposes regularization in the form of K-sparsity on the individual WH transform coefficients—namely, α_S in Equation 1 and applies equal regularization strength across all terms of all degrees [2, 10]. However, this inductive bias counters prevailing beliefs that many sequence functions of interest in biology (and beyond) are likely governed by Occam's razor: that simpler functions—here, those with lower-degree epistatic terms—are more the rule, while higher-degree epistasis are more the exception. For example, WH spectral decomposition of the fitness function of the *Entacmaea quadricolor* fluorescent protein [17] shows that 87% of the energy in WH space is contained in degree 1 and 2 terms, and $> 99\%$ is contained within the first five degrees. A.3

Consequently, our goal is to instead impose regularization at the level of epistatic degree, decoupling the strength of the regularization at each degree. Specifically, we add a regularization loss to our neural network loss function that encodes these beliefs. In its most general form, our regularization loss takes the form:

$$\mathcal{L}_{reg}(\alpha) = \sum_{d=2}^{L} \lambda_d \|\alpha_d\|_2^2, \quad \text{where} \quad \alpha \in \mathbb{R}^{2^L} \equiv (\alpha_d)_{d\in\{0,1,\ldots L\}} \tag{4}$$

and contains $d-1$ hyper-parameters. The reader may notice we have omitted $d = \{0, 1\}$, which do not get regularized. These represent respectively the bias effect, and the "single-site" effects, both of which we should be well-powered to estimate, and hence leave unregularized. The $\|\alpha_d\|_2$ terms are sometimes referred to as the "energy" for degree d [2] and can also be interpreted as the amount of variance explained by each epistastic degree in Equation 3. Because it has been observed that the decay in energy by epistatic order follows a roughly exponential rate [7], we correspondingly assign each $||\alpha_d||$ an exponentially growing penalty. Specifically,

$$\lambda_d = \lambda_0 e^d \quad \text{where} \quad d \in \{2, ..., L\} \tag{5}$$

Where λ_0 is a hyperparameter that controls the overall regularization strength across all epistastic degrees. One could also adjust the base of this exponent to better match specific domain knowledge. Because e^d grows exponentially and in natural fitness functions $||\alpha_d||_2^2$ decays exponentially with respect to d, latter terms in Equation (4) become very sensitive to rounding error. To avoid this, we lump all terms past $d = 5$ into a single term, resulting in the following:

$$\mathcal{L}_{reg}(\alpha) = (\sum_{d=2}^{5} e^d \|\alpha_d\|_2^2 + e^6 \sum_{d=6}^{L} \|\alpha_d\|_2^2), \tag{6}$$

Where we've dropped λ_0. We'll see it reappear in 3.3 as a regularization strength hyperparameter

3.3 Connecting the regularization and neural network losses

Applying the desired regularization, specified in epistatic degree "currency" ($\{\alpha_d\}$), to a neural network, $g(x; w)$, with "currency" of parameter weights (w), will require some creativity because of the mismatch in these currencies. Let us start by simply writing the overall loss in these mismatched currencies. For the neural network, we assume a Gaussian likelihood for the predictive model, $p(y|x)$, with homoscedastic noise, which reduces to the familiar Mean Squared Error (MSE) loss for estimating the parameter weights, w. Together with our regularization loss, the overall loss is thus

$$\mathcal{L}(w) = \sum_i (g(x_i, w) - y_i)^2 + \lambda_0 \mathcal{L}_{reg}(\alpha), \tag{7}$$

where the training data are denoted, $D = \{(x_i, y_i)\}$. However, given the currency mismatch, as written, we have no way to optimize it because it is unclear how to compute $\frac{\partial \mathcal{L}(w)}{\partial w}$, given that we do not know how to compute $\frac{\partial \alpha_d}{\partial w}$. We will need to first be able to compute the mapping $\alpha_d = \alpha_d(w)$, discussed next.

The naive way to compute $\alpha_d = \alpha_d(w)$ would be to use a WH transform of $g(x; \hat{w})$, and then to add up the squared L_2 norms of the WH coefficients of degree d to obtain $\alpha_d(\hat{w})$. However, since our desired regularization does not require the individual coefficients, it turns out that we can instead directly estimate $\{\alpha_d\}$ using the kernel trick, and in particular the ANOVA kernel described above, by way of GP regression. That is, we will estimate $\alpha_d(\hat{w})$ directly through a GP regression, without ever going into the WH basis.

In particular, if we assume each $\alpha_d \sim \mathcal{N}(0, \sigma_d^2)$ and n samples of the inputs and outputs of our neural network, $\{(x_i, \hat{y} = g(x_i; \hat{w})\}_{i=1}^n$, we can stack all these inputs and outputs into matrices, $(X, \hat{Y})$, a GP naturally arises with the model likelihood:

$$\Pr(\hat{Y}|X) \sim \mathcal{N}(\hat{Y}; 0, \sum_{d=0}^{L} \sigma_d^2 \bar{K}_d) \tag{8}$$

where $\bar{K}_d$ is the Kernel matrix on the training data, X, using the d^{th}-degree ANOVA kernel function, $K_d(x, x')$ that has been normalized to have ones on the diagonal, and $\{\sigma_d^2\}$ are free parameters to be estimated. Evidently, any good estimate of $\{\sigma_d^2\}$ must also be a good estimate of $\{\|\alpha_d\|_2^2\}$, since $\mathbb{E}[\|\alpha_d\|_2^2] = \sigma_d^2$.

One can estimate these $\{\sigma_d^2\}$, in Equation 8 using either MLE, or Method of Moments (MoM) which matches the covariance to the empirical covariance. The solution to the MoM loss function has a closed form that can be computed in linear time and is thus immediately differentiable with respect to the weights of the neural network, w—for which the dependency comes through $\hat{\sigma_d^2} = \hat{\sigma_d^2}(\hat{w})$, and which is required to optimize our main loss function in Equation 7 (see A.4 for details). However, MoM is known to be statistically less efficient, here requiring samples covering nearly the entire domain of g to provide an accurate parameter estimate. In contrast, obtaining the MLE solution is computationally expensive, naively scaling cubically with the number of training data; however, MLE, yields good estimates with far fewer samples. Also, as solving for the MLE solution is an iterative procedure requiring gradient-descent, the solution is not trivially differentiable with respect to the weights of the neural network. This poses a challenge in computing $\frac{\partial \mathcal{L}(w)}{\partial \alpha_d}$, required to optimize our overall loss in Equation 7.

Given all possible samples, $\{(x_i, \hat{y} = g(x_i; \hat{w})\}$, both MoM and MLE can perfectly estimate the neural network epistatic-degree energies, $\hat{\sigma_d^2} = ||\alpha_d||_2^2$. For any sub-sample, the estimates will be exact, with MLE providing the more accurate estimates for a given sub-sample size. Similarly to [10], the number of such samples will also provide us with a knob we can turn to gracefully trade off accuracy of the estimate with speed of the computation. Notably, as in Gorji *et al.*, the tradeoff in accuracy is only for the *regularization* loss, not the neural network loss itself. We can now update our overall loss to reflect the fact that we have tied together the currencies, such that each component is now a function of neural network parameters, w:

$$\mathcal{L}(w) = \sum_i (g(x_i, w) - y_i)^2 + \lambda_0 \sum_{d=2}^{L} e^d ||\hat{\sigma}_d^2(w)||_2^2, \tag{9}$$

Because MLE is statistically more efficient, it is our preferred method of estimation, although for some small-scale experiments we make use of MoM because of its speed. However, the issue is that when using MLE for the GP estimation, the mapping between w and $\hat{\sigma}_d^2$ is only implicit, because of MLE's iterative gradient descent procedure. Consequently, the required derivative, $\frac{\partial \alpha_d}{\partial w}$ is not readily computable. Although in principle one could let modern day auto-differentiation unroll

the computation graph to calculate the required meta-model loss function gradient, this would be prohibitively computationally expensive. Instead, we can rely on the implicit function theorem to construct an "implicit layer" [3, 5, 14] that abstracts away the inner optimization problem to avoid differentiating through the unrolled inner optimization steps.

Roughly speaking, the way this works is that the forward pass of back-propagation is as one might expect—that is, one optimizes the density loss. However, the backward pass only differentiates at a fixed point of the density loss, namely, when the iterative solver to minimize the density loss remains unchanged (*e.g.*, the gradient is zero and does not change). This is accomplished by use of the implicit function theorem. Further details can be found in [1].

3.4 Larger alphabets and generalization of ANOVA kernel

Our narrative started with the use of only binary sequence features for the regularization, as is required by previous approaches [10, 2], and continued as such for clarity and intuition. However, one of the advantages of our approach is that we can readily generalize our to non-binary alphabets to natively accommodate nucleotide and amino acids. With a few key adjustments, all of the aforementioned methodological development can be made to generalize to alphabets of any size, by way of the Graph Fourier basis, a natural extension of the WH basis [6][26][24]. The intuitive outcome of using this generalized basis is that each amino acid, for example, gets represented with a vector of length $20 - 1 = 19$, using an encoding that preserves certain symmetries that are required for Graph Fourier analysis. Note we use vectors of length 19 to represent 20 tokens: this helps us avoid colinearity. We make use of this same basis herein, the details of which are in A.1.

3.5 Putting it all together: EDR-NN

We have now progressively described all the moving parts: the desire to regularize a neural network by epistastic degree; how to estimate the variance accounted for by each epistastic degree in a given neural network with the help of a GP regression trained on inputs and corresponding outputs from the neural network; and the need for implicit layer machinery to effectively compute the gradient of our regularized neural network loss with respect to the GP regression model which yields a nested set of optimizations—the GP optimization inside of the regularized neural network loss optimization. Combining all of these yields our EDR-NN, for which there is an algorithm overview in the A.5. Having written down our overall loss function and figured out how to efficiently take gradients of it, our training is otherwise standard.

Code availability With written permission from the Program Chair for RECOMB 2024 (Jian Ma), we submit our code as Supplementary Information rather than a GitHub repository. Upon acceptance, we will publicly release the code.

4 Experimental Results

We performed two main sets of experiments. In the first, we construct a synthetic data set to investigate how different inductive biases interact with i) the label noise level and ii) the size of the training data. The second set of experiments are on real experimental data, where we compare the performance of EDR-NN to the the WH-L_1-regularization of Gorji *et al.*, dubbed HashWH, and to a standard neural network. For all experiments, we followed similar guidelines to determine our neural network architecture, on which we applied both EDR-NN and HashWH: Assuming a sequence of length L, we used a network architecture of $(\textit{input shape}) \times (nL) \times (nL) \times L \times 1$, where n is a multiplier set equal to 10 for real data and 20 for our simulated dataset to foster faster convergence. These are the same architecture guidelines as [10] and [2], adjusted to handle non-binary sequences as well. All methods were implemented using Jax for more efficient implicit differentiation and trained and evaluated on a combination of Nvidia GPUs with 10GB, 24GB, and 48GB of memory.

4.1 Characterizing the effects of inductive bias with simulated data

To simulate a ground-truth fitness function which maps sequence to protein property, we follow Busia *et al.*, describing only a high-level overview here [7]. We used a linear function with randomly generated epistatic effects—random in which effects, how many effects, and what their values are. Epistatic effects of all orders were allowed, but the specific ones were sampled randomly based on the empirical distribution of such effects observed for the *Entacmaea* quadricolor fluorescent protein [17], combined with the observation that number of contacts in a protein scales linearly with sequence length [4, 27], which has been shown to inform on the epistastic landscape [6]. Our synthetic sequences comprised all 2^{10} binary sequences of length 10. We labeled each of these with the simulated fitness function and then added homoscedsatic Gaussian noise with variance, σ^2 to simulate noisy, observed property values. We ran experiments with values of $\sigma^2 \in \{1, 4, 9, 16, 25\}$. For reference, the empirical variance of the simulated, noise-free ground truth labels was 11.04.

We compare and contrast three modeling approaches, all using one identical architecture, differing only in how that architecture was regularized. One was not regularized, "Standard NN", one was regularized using "HashWH" [10], and one using "EDR-NN". HashWH has a tunable parameter b that trades off speed with exactness, and we set this parameter such that HashWH is exact (*i.e.*, $b = 10$ which yields 2^{10} hash bins such that no collisions can occur). Similarly, for EDR-NN, we use MoM estimation for the GP regression with all the $2^{10} = 1024$ possible sequences. For training data, we used 500 sequences and labels chosen at random from the 1024, with the remaining 524 used as test data.

First we examine the predictive accuracy as a function of label noise. As we increased the noise variance added to the noise-free fitness labels, the performance of all methods goes down, as expected. More interestingly, the unregularized model goes down fastest, then HashWH goes down next fastest, followed lastly by EDR-NN(Figure 1a). Thus, when data are generated with the distribution of epistatic interactions based on the epistatic fitness simulation of Busia *et al.*, then our method provides more robustness against noisy labels. Experiments on the real data will shed some light on how realistic that assumption is.

Next we examined the recovery of the epistatic energy landscape as a function of label noise, for each of the methods. We see that for all methods, the recovery becomes harder and harder with increasing noise, but that similar patterns to predictive accuracy occur, with EDR-NN being the most robust noise, followed by HashWH, followed by the unregularized model (Figure 1b). A comparison of the full degree-wise energy landscape is in A.6

4.2 Experiments on real protein data

We compare and contrast the performance of same three methods as in the previous section on real datasets. Additionally, we performed Gaussian Process regression and kernel ridge regression with $\alpha = 0$, and standard L_2 regression also referred to as Ridge regression as baselines with decreasing levels of complexity. For Gaussian Process regression, we considered a linear combination of generalized ANOVA kernels up to degree 5 plus a noise term as the covariance matrix, with the coefficients treated as hyperparameters, selected by maximizing the marginal log-likelihood $\log(\Pr[y \mid X])$ with respect to the coefficients. For the kernel ridge regression, we use a sum of all ANOVA kernels from degree 0 through to 10 to ensure that all significant terms were considered, resulting in a completely unregularized model.

We apply these 5 methods to four protein data sets: fluorescent protein in *Entacmaea* [17]; GFP from [20], AAV Capsid from [23], and Yeast his3 from [18]. More details on the selection process of these dataset are in A.7.

From each protein, we set aside $1,000$ sequence-label pairs for testing, and from the remainder, chose $N \in \{20, 40, 80, 160, 320\}$ sequence-label pairs for training. We re-sampled these training samples 10 times over which we could assess our results for variability. All data were used at the amino acid level (rather than nucleotides).

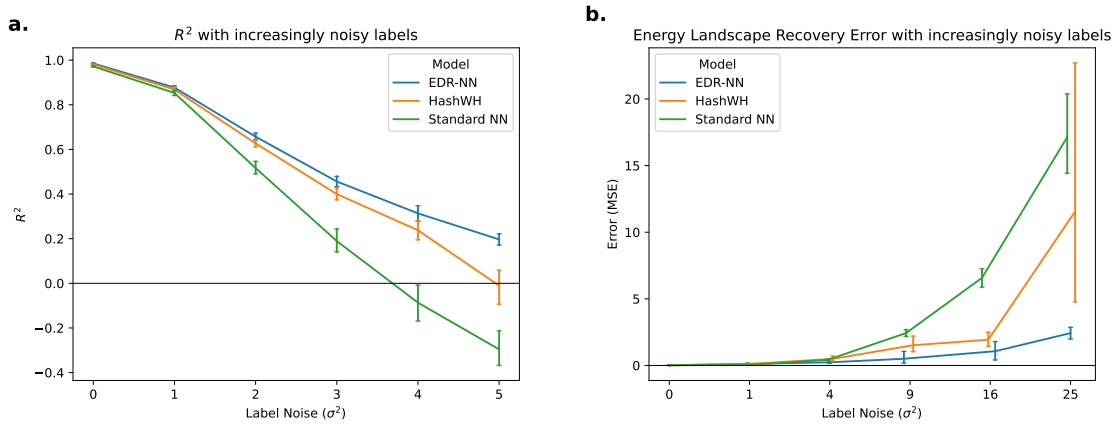

Fig. 1: **Simulated experiments: a.**, Test set prediction accuracy, measured by R^2, as a function of label noise. **b.**, Recovery of epistastic energies at each degree, as a function of label noise. X-axis has been randomly jittered to help avoid overlapping.

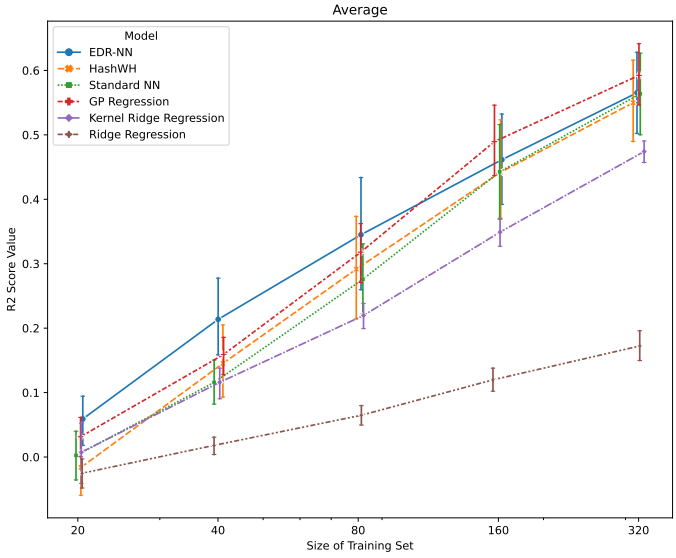

Fig. 2: **Average test set prediction accuracy,** measured by R^2, across all datasets. Error bars are 95% confidence intervals on the sample mean, computed by bootstrapping. Sizes have been randomly jittered to minimize overplotting. In both plots, 95% confidence intervals around the sample mean were generated by bootstrapping

For each training data set size N, we performed hyperparameter selection for the learning rate and batch size on only the unregularized neural network, and then used this setting for all of the methods. The number of epochs setting was set to that maximum used for the optimal batch and learning rate, and then all three of these parameters were used for all further experiments. We did not

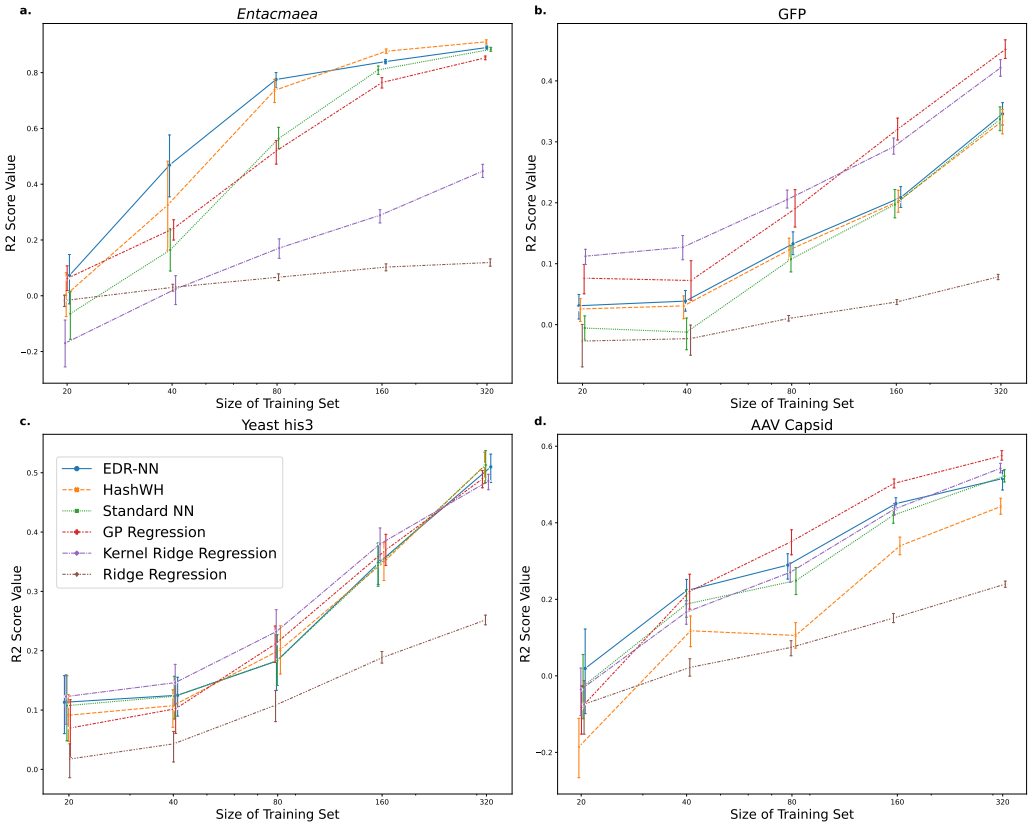

Fig. 3: R^2 **vs training set size** for **a.**, *Entacmaea*, **b.**, GFP, **c.**, his3, and **d.**, AAV Capsid. Error bars are 95% confidence intervals around the sample mean computed using bootstrapping, and the legend is consistent across all figures.

perform architecture search, instead relying on the same heuristic as related papers, which precisely prescribes a neural network to use $[10, 2]$.

The regularization for HashWH and EDR-NNstrength were each set by a separate hyperparameter selection sweep, using the already set learning rate, batch size and maximum number of epochs as just described. For HashHW, we searched over $\alpha = \{0.0001, 0.001, 0.01, 0.1\}$, the same values as used in their work. For EDR-NN, we also searched over $\alpha = \{0.0001, 0.001, 0.01, 0.1\}$. We did similar sweeps over larger ranges for ridge regression and Kernel Ridge Regression, but interestingly Kernel Ridge always performed best with $\alpha = 0$, meaning it performed best when completely unregularized. Both HashHW and EDR-NNonly approximately estimate the needed epistatic coefficients/energies for their respective regularization. For HashHW, the degree of approximation is dictated by the hyperparameter b, which yields 2^b hash bins, and also dictates that 2^b neural network samples are needed to estimate their approximate WH transform with each gradient step of the neural network. They sweep through $b = \{7, 10, 13, 16\}$, and note that generally, the larger b, the better the performance. Since either our GPU or computer ran out of memory when we selected $b = 16$, we set $b = 13$ for all experiments. Gorgji *et al.*select 16 only for on the GFP data set. For EDR-NN, the degree of approximation is dictated by the number of samples used to estimate the parameters of the GP regression, which we fixed at 500, which constitutes $\frac{500}{|\mathcal{A}|^L}$ of the total number possible. We used MLE to estimate the parameters of the GP regression model. All method-dependent hyperparame-

ters were selected, for each N, using 5 train-validation splits of the training data (*i.e.*not touching the $1,000$ test points originally held out).[1]

One final note is that the method of Gorji *et al.*/ (and similarly Aghazadeh), can only work natively with binary alphabets. In those papers, they consequently had to binarize their data if it was not already in binary form. For example, for GFP, instead of allowing all twenty amino acids, they converted the data into "wild type" amino acids, vs not. Since this did not seem a realistic endeavour, herein, we instead used a one-hot encoding, thereby retaining information for all amino acid values.

Results on real protein data sets Averaging the results across all four data sets, we see the trend that EDR-NN performs maximally for small datasets, but is surpassed by the GP regression for larger datasets. HashWH lags behind both but is a close contender, followed by kernel ridge regression (Fig. 3) and ridge regression. Looking more closely at the invidual protein results, we observe that GP regression and kernel ridge regression substantially outperform the other methods for GFP, with EDR-NN and HashHW largely tied below. For AAV Capsid and Entacmeae, EDR-NN performs best for the smaller training data set sizes, and then performs more on par with HashWH and worse than the GP regression for larger training data sizes. Finally, on Yeast his3, the kernel ridge regression with seems to perform best by a small margin, with the other methods tied, other than ridge regression that lags behind. (Fig. 2)

5 Discussion

We developed a new method to regularize neural networks in a manner that allows for different regularization strength for each epistatic order, in an L_2-norm sense. The flexibility of this regularization framework enabled us to explore regularization of protein fitness models to more strongly regularize higher order epistatic terms, as a group, by way of an ANOVA kernel as a regularizer to a neural network. We showed that under plausible simulation scenarios of epistatic landscapes, that our approach behaves sensibly, and outperforms other models that encode different inductive biases. On real data, we find that overall, our model may provide an improvement, but that kernel methods using our generalized ANOVA kernel can perform quite well. In fact, GP regression did not perform substantially worse than the standard MLP in any of the datasets and sometimes outperformed it substantially, even beating out the regularized MLPs.

A number of further directions would be worth exploring. First one limiting element of EDR-NN as is, was the cubic time complexity of the GP regression parameter estimation, limiting the accuracy with which we could estimate the epistastic energies. We anticipate that by leveraging existing innovations from the GP community, such as the use of pseudo-points, and so-forth, that we may extract better statistical power yet. We may be able to choose training samples more intelligently, such that they are jointly more informative.

[1] Both Gorji *et al.*[10] and Aghazadeh *et al.*[2] for reasons that are not clear, use a validation set that is substantially larger (up to 100 times) than the training data, an odd choice that we did not replicate.

References

1. Implicit layer tutorial. http://implicit-layers-tutorial.org, accessed: 2023-11-03
2. Aghazadeh, A., Nisonoff, H., Ocal, O., Brookes, D.H., Huang, Y., Koyluoglu, O.O., Listgarten, J., Ramchandran, K.: Epistatic net allows the sparse spectral regularization of deep neural networks for inferring fitness functions. Nature communications **12**(1), 5225 (2021)
3. Amos, B., Kolter, J.Z.: OptNet: Differentiable optimization as a layer in neural networks. In: Proceedings of the 34th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 70, pp. 136–145. PMLR (2017)
4. Bartoli, L., Capriotti, E., Fariselli, P., Martelli, P.L., Casadio, R.: The pros and cons of predicting protein contact maps. Protein Structure Prediction pp. 199–217 (2008)
5. Blondel, M., Berthet, Q., Cuturi, M., Frostig, R., Hoyer, S., Llinares-Lopez, F., Pedregosa, F., Vert, J.P.: Efficient and modular implicit differentiation. In: Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., Oh, A. (eds.) Advances in Neural Information Processing Systems. vol. 35, pp. 5230–5242. Curran Associates, Inc. (2022)
6. Brookes, D.H., Aghazadeh, A., Listgarten, J.: On the sparsity of fitness functions and implications for learning. Proc. Natl. Acad. Sci. U. S. A. **119**(1) (2022)
7. Busia, A., Listgarten, J.: Mbe: model-based enrichment estimation and prediction for differential sequencing data. Genome Biol **24**, 218 (2023). https://doi.org/10.1186/s13059-023-03058-w
8. Cohen, T., Welling, M.: Group equivariant convolutional networks. In: Balcan, M.F., Weinberger, K.Q. (eds.) Proceedings of The 33rd International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 48, pp. 2990–2999. PMLR, New York, New York, USA (20–22 Jun 2016), `https://proceedings.mlr.press/v48/cohenc16.html`
9. Gelman, S., Fahlberg, S.A., Heinzelman, P., Romero, P.A., Gitter, A.: Neural networks to learn protein sequence–function relationships from deep mutational scanning data. Proceedings of the National Academy of Sciences **118**(48), e2104878118 (2021)
10. Gorji, A., Amrollahi*, A., Krause, A.: A scalable walsh-hadamard regularizer to overcome the low-degree spectral bias of neural networks. In: Conference on Uncertainty in Artificial Intelligence (UAI) (July 2023)
11. Huang, Z., Bai, S., Kolter, J.Z.: Implicit2: Implicit layers for implicit representations. Advances in Neural Information Processing Systems **34**, 9639–9650 (2021)
12. Jing, B., Eismann, S., Suriana, P., Townshend, R.J.L., Dror, R.: Learning from protein structure with geometric vector perceptrons. In: International Conference on Learning Representations (2021), `https://openreview.net/forum?id=1YLJDvSx6J4`
13. Jumper, J.M., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Zídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S.A.A., Ballard, A., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., Back, T., Petersen, S., Reiman, D.A., Clancy, E., Zielinski, M., Steinegger, M., Pacholska, M., Berghammer, T., Bodenstein, S., Silver, D., Vinyals, O., Senior, A.W., Kavukcuoglu, K., Kohli, P., Hassabis, D.: Highly accurate protein structure prediction with alphafold. Nature **596**, 583 – 589 (2021)
14. Lorraine, J., Vicol, P., Duvenaud, D.: Optimizing millions of hyperparameters by implicit differentiation. In: Chiappa, S., Calandra, R. (eds.) Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 108, pp. 1540–1552. PMLR (26–28 Aug 2020)
15. Notin, P., Dias, M., Frazer, J., Hurtado, J.M., Gomez, A.N., Marks, D., Gal, Y.: Tranception: protein fitness prediction with autoregressive transformers and inference-time retrieval. In: International Conference on Machine Learning. pp. 16990–17017. PMLR (2022)
16. Pazokitoroudi, A., Wu, Y., Burch, K.S., Hou, K., Zhou, A., Pasaniuc, B., Sankararaman, S.: Efficient variance components analysis across millions of genomes. Nature communications **11**(1), 4020 (2020)
17. Poelwijk, F.J., Socolich, M., Ranganathan, R.: Learning the pattern of epistasis linking genotype and phenotype in a protein. Nature communications **10**(1), 4213 (2019)
18. Pokusaeva, V.O., Usmanova, D.R., Putintseva, E.V., Espinar, L., Sarkisyan, K.S., Mishin, A.S., Bogatyreva, N.S., Ivankov, D.N., Akopyan, A.V., Avvakumov, S.Y., et al.: An experimental assay of the interactions of amino acids from orthologous sequences shaping a complex fitness landscape. PLoS genetics **15**(4), e1008079 (2019)

19. Rasmussen, C.E., Williams, C.K.I.: Gaussian processes for machine learning. Adaptive computation and machine learning, MIT Press (2006)
20. Sarkisyan, K.S., Bolotin, D.A., Meer, M.V., Usmanova, D.R., Mishin, A.S., Sharonov, G.V., Ivankov, D.N., Bozhanova, N.G., Baranov, M.S., Soylemez, O., et al.: Local fitness landscape of the green fluorescent protein. Nature **533**(7603), 397–401 (2016)
21. Shawe-Taylor, J., Cristianini, N.: Basic kernels and kernel types. In: Kernel Methods for Pattern Analysis, pp. 291–326. Cambridge University Press (Jun 2004)
22. Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press, Cambridge, UK (2004)
23. Sinai, S., Jain, N., Church, G.M., Kelsic, E.D.: Generative aav capsid diversification by latent interpolation. bioRxiv pp. 2021–04 (2021)
24. Stadler, P.F., Seitz, R., Wagner, G.P.: Population dependent fourier decomposition of fitness landscapes over recombination spaces: Evolvability of complex characters. Bulletin of Mathematical Biology **62**, 399–428 (2000)
25. Staller, M.V., Holehouse, A.S., Swain-Lenz, D., Das, R.K., Pappu, R.V., Cohen, B.A.: A high-throughput mutational scan of an intrinsically disordered acidic transcriptional activation domain. Cell systems **6**(4), 444–455 (2018)
26. Stormo, G.D.: Maximally efficient modeling of dna sequence motifs at all levels of complexity. Genetics **187**(4), 1219–1224 (2011)
27. Vendruscolo, M., Kussell, E., Domany, E.: Recovery of protein structure from contact maps. Folding and Design **2**(5), 295–306 (1997)
28. Wu, N.C., Dai, L., Olson, C.A., Lloyd-Smith, J.O., Sun, R.: Adaptation in protein fitness landscapes is facilitated by indirect paths. Elife **5**, e16965 (2016)
29. Yang, G., Salman, H.: A fine-grained spectral perspective on neural networks (2020)

A Appendix

A.1 Description of the Fourier Basis

To use larger alphabets, we use the generalization of the WH matrix described by [6] (see sections titled *Fourier Bases for Fitness Functions* and *Materials and Methods: Fourier Bases*); we adopt their convention of calling this generalization the Fourier matrix. Like the WH matrix, the Fourier matrix is defined recursively. Let $X \in \mathcal{A}^L$ where $q := |\mathcal{A}|$ is the size of the alphabet, $\mathcal{A}$, and denote

$$P_q := I_q - \frac{2ww^T}{\|w\|_2^2}, \tag{10}$$

where I_q is the $q \times q$ identity matrix, $w := \mathbf{1}_q - \sqrt{q}e_1$, $\mathbf{1}_q$ is the vector of all ones, and e_1 has its first element set to one and the rest zeros. Each row of P_q corresponds to a different length-1 sequence. The first column is constant for all rows, while the remaining $q-1$ elements uniquely identify an alphabet element. Specifically, if we write

$$P_q = \frac{1}{\sqrt{q}} \left[\mathbf{1}_q \mid P_q' \right], \tag{11}$$

where $|$ denotes column-wise concatenation, then each row of P_q' constitutes a length-$(q-1)$ encoding of an element of the alphabet. We will let $p_q(a)$ denote the row of P_q' that encodes alphabet element a, which can be assigned arbitrarily. To define F_L, the Fourier matrix for sequences of length L with alphabet size q (we will ignore dependence on q in the notation for simplicity), we set $F_1 := P_q$ and

$$F_L = F_{L-1} \otimes P_q, \tag{12}$$

where $\otimes$ denotes the Kronecker product. Note that $\Phi_L = H_L$ when $\mathcal{A} = \{-1, 1\}$. As in the binary alphabet case, each row of the Fourier matrix contains all possible epistatic interactions of a sequence. Note that while in the binary case, an order-d epistatic term of the sequence X is encoded by a single scalar, $\prod_{i \in S} X_i \in \{1, -1\}$, in the case of larger alphabets, an order-d epistatic term is encoded by a vector of length $(q-1)^d$. In particular, the vector encoding the interaction between the sites in set S is given by

$$F_L(S) = \frac{1}{\sqrt{q^L}} \bigotimes_{i \in S} p_q(X_i). \tag{13}$$

A row of the Fourier matrix is the concatenation of these encodings for all possible sets, $S \subseteq [L]$.

This generalized encoding preserves the property that any function from a sequence can be represented as a linear function of the generalized feature embedding. Namely, for any fitness function $f : \mathcal{A}^L \to \mathbb{R}$ can be represented uniquely as a sum of linear functions of the embeddings of all possible subsets:

$$f(X) = \sum_{S \subseteq \mathcal{A}^L} \vec{\alpha}_S \cdot F_L(S), \tag{14}$$

where $\vec{\alpha}_s \in \mathbb{R}^{(q-1)^{|S|}}$.

A.2 Proof of correctness fast ANOVA kernel

First, we start by proving that $K_d(x, x')$ has a closed form solution with respect the he hamming distance, H between x and x'

Proof. Note that for each $S \subseteq [L]$ such that $|S| = d$, S can include anywhere from 0 to $\min(d, H)$ "disagreeing" indices, that is indices where x disagrees with x'. Let $\Delta_S = |\{i \in S | x_i \neq x'_i\}|$ be the number of disagreeing indices in S. Now we have:

$$\prod_{i \in S} x_i \cdot \prod_{i \in S} x'_i = \prod_{i \in S} x_i x'_i$$

And since $x_i x'_i = 1$ if $x_i = x'_i$ and -1 otherwise,

$$\prod_{i \in S} x_i x'_i = (-1)^{\Delta_S}$$

Now,

$$\begin{aligned} K_d(x, x') &= \sum_{|S|=d} \prod_{i \in S} x_i x'_i \\ &= \sum_{i=0}^{\min(d,H)} \left| \left\{ S \in [L] \;\middle|\; |S| = d, \Delta_S = i \right\} \right| \cdot (-1)^i \\ &= \sum_{i=0}^{\min(d,H)} \binom{L-H}{d-i} \binom{H}{i} \cdot (-1)^i \\ &= \sum_{i=0}^{d} \binom{L-H}{d-i} \binom{H}{i} \cdot (-1)^i, \end{aligned}$$

Since $\binom{H}{i} = 0$ if $i > H$. □

Because this is solely a function of H, and otherwise independent of the elements of X and X', we can do a single $O(Ld)$ (which is the cost of running *each* ANOVA kernel computation with the standard computation method) pre-computation step to compute the term $\sum_{i=0}^{d} \binom{L-H}{d-i}\binom{H}{i} * (-1)^i$ for each $H \in \{0, ..., L\}$, and then every subsequent kernel computation can be computed $O(L)$ time, since for any given x, x', we only need to compute $|x - x'|_H$, and then perform a constant-time retrieval operation. If we would like to compute the kernel for multiple values of d, the cost becomes $O(L + d)$.

Now, we will prove an almost identical claim for the general Fourier Basis described in A.1 We would like to compute

$$K_d(X, X') = \langle \Phi_d(x), \Phi_d(x') \rangle \tag{15}$$

where Φ_d are defined almost identically as in the binary case:

$$\Phi_d(x) = (F_S)_{S \subseteq [L], |S|=d} \tag{16}$$

Except these Kronecker-product defined F_S embeddings replace the monomials we saw in the binary case. To compute the inner product of this featurization for two sequences, we can again leverage the fact that the inner product is solely defined by the hamming distance between the two sequences.

Claim. For $\forall X, X' \in \mathcal{A}^n$ such that $|x - y|_H = H$,

$$K_d(x, x') = \frac{1}{q^L} \sum_{i=0}^{d} \binom{L-H}{d-i} \binom{H}{i} \left(\frac{q-1}{q} \right)^{d-i} \left(-\frac{1}{q} \right)^i \tag{17}$$

Proof. First, we must note that

$$P_q P_q^T = (I_q - \frac{2ww^T}{\|w\|_2^2})(I_q - \frac{2ww^T}{\|w\|_2^2}) \tag{18}$$

$$= I_q - 2\frac{2ww^T}{\|w\|_2^2} + \frac{2ww^T}{\|w\|_2^2}\frac{2ww^T}{\|w\|_2^2} \tag{19}$$

$$= I_q - 4\frac{ww^T}{\|w\|_2^2} + 4\frac{ww^T ww^T}{\|w\|_2^4} \tag{20}$$

$$= I_q - 4\frac{ww^T}{\|w\|_2^2} + 4\frac{ww^T}{\|w\|_2^2} \tag{21}$$

$$= I_q \tag{22}$$

And since each row of P_q corresponds to $\left[\frac{1}{\sqrt{q}} \mid p_q(a)\right]$ for an $a \in \mathcal{A}$, the $\forall a, b \in \mathcal{A}^L$

$$\frac{1}{\sqrt{q}} \cdot \frac{1}{\sqrt{q}} + p_q(a) \cdot p_q(b) = \begin{cases} 1, & \text{if } a = b \\ 0, & \text{otherwise} \end{cases} \tag{23}$$

$$\implies p_q(a) \cdot p_q(b) = \begin{cases} \frac{q-1}{q}, & \text{if } a = b \\ \frac{1}{q}, & \text{otherwise} \end{cases} \tag{24}$$

Now let $S \subseteq [L]$, and once again, let $\Delta_S = |\{i \in S | X_i \neq X_i'\}|$ be the number of disagreeing indices in S.

$$\left\langle \frac{1}{\sqrt{q^L}} \bigotimes_{i \in S} p_q(x_i), \frac{1}{\sqrt{q^L}} \bigotimes_{i \in S} p_q(x_i') \right\rangle \tag{25}$$

$$= \frac{1}{q^L} \left\langle \bigotimes_{i \in S} p_q(x_i), \bigotimes_{i \in S} p_q(x_i') \right\rangle \tag{26}$$

$$= \frac{1}{q^L} \bigotimes_{i \in S} \langle p_q(x_i), p_q(x_i') \rangle \text{ by the mixed product prop.} \tag{27}$$

$$= \frac{1}{q^L} \left(\frac{q-1}{q}\right)^{d-\Delta_S} \left(-\frac{1}{q}\right)^{\Delta_S} \tag{28}$$

And we can repeat the counting argument from the binary case to get the desired summation. □

Clearly, this generalized kernel computation has the same runtime as the binary case,

A.3 Energy Landscape of *Entacmaea*

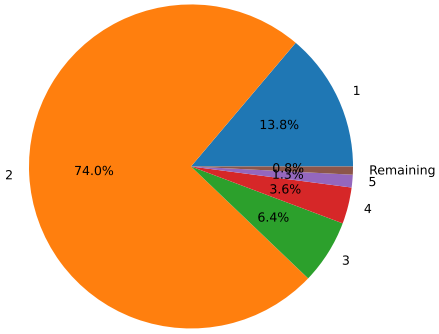

Fig. 4: Percent of Energy Explained by Each Degree of Epistasis in *Entacmaea*

This figure is generated using code from [10] used to generate **Fig 4.d**, except observing all epistatic interactions instead of just the top 100.

A.4 MoM Closed Form Solution

Estimating $\sigma_\epsilon^2, \sigma_d^2$ such that $y \sim N(0, \sum_{d=0}^{L} \sigma_d^2 K_d + \sigma_\epsilon^2 \mathbb{I})$ is actually a common problem in heritability estimation. See [16] for more background. Unlike in GP literature, heritability estimates are frequently done with Method of Moments, partially because they do not necessarily assume a Normal distribution. We can start with Equation 7 of [16]:

$$\begin{bmatrix} T & b \\ b^T & N \end{bmatrix} \begin{bmatrix} \sigma^2 \\ \sigma_e^2 \end{bmatrix} = \begin{bmatrix} c \\ y^T y \end{bmatrix} \tag{29}$$

Where T is a $d \times d$ matrix with entries $T_{k,l} = tr(\bar{K}_k \bar{K}_l)$ and $c \in \mathbb{R}^d$ is defined so that $c_d = y^T K_d y$. We can ignore all other terms because we are in a noiseless regime, so σ_e is 0.

We know $K_k = \phi(X)_k^T \phi_k(X)$, so

$$tr(\bar{K}_k \bar{K}_l) = \frac{1}{\binom{L}{k}\binom{L}{l}} tr(\phi_k(X)^T \phi_k(X) \phi_l(X)^T \phi_l(X)) \tag{30}$$

And since each ϕ_d is a submatrix of an orthogonal ϕ matrix, if $k \neq l$ then $tr(\bar{K}_k \bar{K}_l) = 0$ and 1 if $k = l$, which leaves us with $\sigma^2 = c$, which implies $\sigma_d^2 = c_d = y^T K_d y$ solves our normal equation, and we are done. One can easily verify by similarly decomposing the y and K_d explicitly into $y = \phi(X)\alpha$ and $K_d = \phi_d(X)^T \phi_d(X)$ that if we have all samples, then $c_d = y^T K_d y = \|\alpha_d\|_2^2$, since then $\phi(X)$ is a full orthogonal matrix.

A.5 Complete Algorithm

Algorithm 1: EDR-NN

```
def EDR-NN(model, find_sigmas, lr, data_loader, reg_alpha, num_epochs):
    model.init() ;
    optimizer = Adam(lr) ;
    for epoch in num_epochs do
        for X_train, y_train in data_loader do
            y_preds = model(X_train) ;
            mse_loss = MSE(y_train, y_preds) ;
            mse_grads = grad(mse_loss) ;
            alpha_preds = find_alphas(model, method = MLE) ;
            reg_loss = sum([e^d * alpha_preds[d] for d in range(2, 6)]);
            reg_grads = grad(reg_loss) ;
            optimizer.update(model, mse_grads + reg_alpha * reg_grads);
        end
    end
    return model;
def find_sigmas(model):
    if method == 'MLE' then
        X_sample = sample_from_domain() ;
        Kernels = compute_kernels(X_sample, X_sample);
        y_sample = model(X_sample);
        return argmax_σ(y_sample, Kernels);
    end
    else if method == 'MoM' then
        X_sample = sample_entire_domain() Kernels = compute_kernels(X_sample, X_sample);
        y_sample = model(X_sample);
        return y_sample.T @ Kernels @ y_sample * (alphabet_size ** sequence_length;
    end
def MLE_objective(sigmas, y_sample, Kernels):
    Cov = sum([s_d * K_d for s_d, K_d in zip(sigmas, Kernels)];
    return GaussianLogLikelihood(y_sample; 0, Cov)
```

A.6 Comparison of degree-wise energy landscapes of different models on simulated data

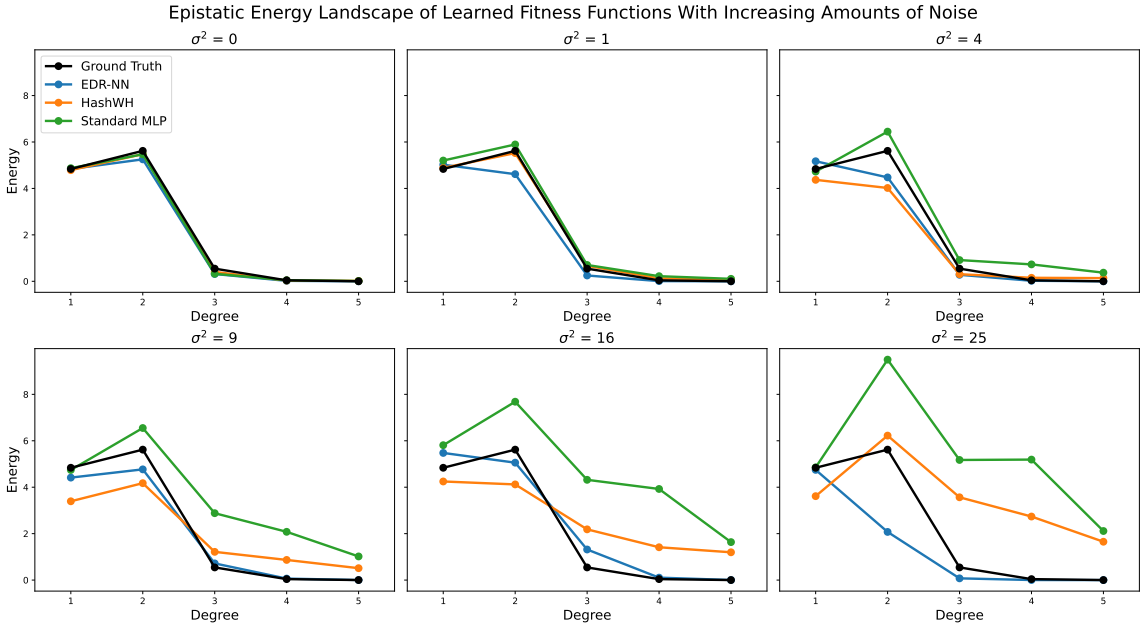

Fig. 5: **Degree-wise energy landscape of each model.**
Closer to black line is better.

Note that as the variance of noise increases, HashWH and the Standard MLP hallucinate higher order interactions, while EDR-NN does a much better job of staying faithful to the ground truth energy landscape, even when the signal to noise ratio is quite poor.

A.7 Dataset Selection and Curation

We considered all datasets in ProteinGym [15] that had assay-labelled data on sequences with more than 4 mutations away from wildtype, so as to have access to complex epistatic landscapes, of the kind we anticipate will be more routine than those comprising single, or single and some pairwise mutations. There were only four such datasets: GFP [20], Yeast his3 (called his7 in ProteinGym) [18], AAV Capsid [23], and Yeast GCN4 [25]. Of these four, we chose not to work with GCN4 because in all our our training set sizes, all methods were performing with $R^2 < 0.1$.

We also considered all of the real protein datasets used on [10]: *Entacmaea*, a combinatorially complete dataset on four positions of GB1 [28], and a binarized form of the GFP dataset, where each position is converted to a 1 if that position is mutated from wildtype and 0 otherwise. Duplicate sequences have their fluorescence values averaged. We omitted the GB1 dataset because the four positions selected for combinatorial assaying were handpicked to be significant, which makes epistatic interactions between those positions to be far more likely than they would be had the positions been selected uniformly at random. As a result the "ground truth" fitness function derived from those four positions will not observe the same strong decaying rate of epistatic energy as a natural fitness function would, which our method assumes. We omitted the modifed GFP dataset because we are including the complete GFP dataset in our experiments, and because when the dataset is binarized

as such, it becomes relatively uninteresting, since averaging across all mutations in each position neutralizes local extrema that arise specific mutations (as opposed to the presence of a mutation), which are exactly the sequences of interest. We truncated each dataset around the DMS range, or the smallest contiguous range of the protein sequence containing all mutations in the dataset, since anything outside of that range is effectively a constant term that cannot inform learning.