Neural Pairwise Regression

Jackson Burns

Agenda

Regressing Hard Targets

Canonical Regression Problem

- Many fields seek **surrogate models** for arbitrary quantities of interest
  - solubility as a function of a molecular embedding instead of experiment
  - band gap as a function of formula instead of expensive simulation
- When target y is continuous this is a regression problem $y = f(x; \theta)$
  - x is some static or learned *embedding* i.e. a vector of scalars
  - $f(\ldots; \theta)$ is an arbitrary model with parameters

Regression Methods

- Terminology is loose[1] but broadly we use:
  - Statistical Modeling (SM): (regularized) linear methods
  - Machine Learning (ML): Decision Trees, Neighbors-Based, and *Ensembles*
  - Artificial Intelligence (AI): Neural Networks (NNs)
- Broadly speaking, moving down that list:
  - requires more data to fit an accurate model because *f* has more parameters θ
  - offers a higher accuracy 'ceiling' and can fit 'harder' datasets
  - reduces interpretability

---

[1]Some use these interchangeably i.e. linear regression = AI

Canonical Neural Network

- NNs are especially popular
  - easy to train with high-level languages
  - trivial hardware acceleration
  - interpretability drawbacks secondary to improved performance
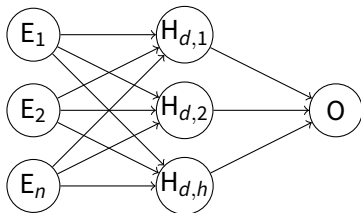- Generality reduces the 'time-to-model'

Figure 1: Schematic of typical Feedforward Neural Network with input encoding E of dimension n, Hidden Layers H of height h and depth d, and Output O. Arbitrary activation functions (i.e. ReLU) at output of each node not shown.

Data Limitations

- Large, high-quality datasets are rare
  - *especially* in science domains where experiments are expensive
  - negative ramifications for coverage of feature space
- Mapping structure → target is *hard*
  - static and learned representations are *very* high-dimensional, $O(100)$+ features
  - relationships are often unpredictably non-linear (i.e. Activity Cliffs, some examples in (Stumpfe et al., 2019))

The Wicked Problem
Neural networks are well-suited for our problems but we lack sufficient data to fit them accurately.

Reformulating Regression

Pairwise Regression

- We can address The Wicked Problem by recasting our regression problem to difference prediction:
  - instead of predicting the output directly: $y = f(x; \theta)$
  - we predict the *difference* in target for two inputs at a time: $y_1 - y_2 = \Delta_{1,2} = f(x_1, x_2; \theta)$
- This addresses our key constraint
  - quadratic increase in the amount of training data
- Offers additional benefits
  - ensemble of predictions during inference provides well-calibrated uncertainty estimates (Wetzel et al., 2020)
  - theoretical evidence that inference is easier (OOS vs OOD) (Netanyahu et al., 2023)

Pairwise Architecture

- Choose the most *basic* approach to achieve this mapping
    - inputs are directly concatenated
    - keeps SHAP-ability (Lundberg & Lee, 2017)
    - leans on NN Universal Approximation Theorem [a]

---

[a] Closed-form width and depth requirements for ReLU networks known (Shen et al., 2022)
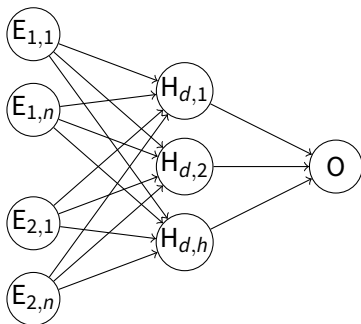

Figure 2: Schematic of a Neural Pairwise Regressor with concatenated input encodings $E_{i,n}$

Need for Naivete

Empirical exploration suggests avoiding **inductive bias** in NN architecture

- Literature has explored alternative design in the general case (see next slide)
- My previous work with collaborators explored it specifically for solubility (Attia et al., 2024)

Existing Alternatives

There are many ways to inject bias into the model:

- (Tynes et al., 2021): input $x_1 \oplus x_2 \oplus (x_1 - x_2)$ and use ML
  - inductive bias in input distance (i.e. unreduced manhattan) is probably unhelpful
- (Netanyahu et al., 2023): include an operation in later layers (i.e. subtraction) to enforce self 'loop'
  - do we really want to enforce this property?
- (Wetzel et al., 2020): additional latent layers for each input
  - makes it *possible* for network to learn a different embedding for each input 'branch' (bad!)
  - (Wetzel et al., 2021) moved away from this and has no embedding layers
  - a *shared* learnable embedding would be OK (foreshadowing)

The Bitter Lesson

Rich Sutton famously pointed out in a blog post [2] that in the Computer Vision world:

- architectures based on human intuition for how to process images failed
- Convolutional NNs (which have almost *no* inductive bias) dominate the space

The Bitter Lesson
The most general, scalable approach is the most effective.

One criticism of this argument is that hardware is no longer following Moore's law … but we (and 99.9% of people) aren't looking at a scale where that matters.

---

[2]Representative publication: (Yousefi & Collins, 2024)

'Physics' Constraints

There are a number of cases of the 'cycle closure' rule that we *might* want to enforce:

- Arbitrary loop: $\Delta_{1,2} + \Delta_{2,n} + \ldots + \Delta_{n,1} = 0$
  - Get this 'for free' with an accurate model
- Pairwise 'loop': $\Delta_{1,2} + \Delta_{2,1} = 0$
  - Requires positional invariance of inputs in f
- Self 'loop': $\Delta_{1,1} = 0$
  - Inductive bias in model architecture (i.e. subtract the latent representations) can explicitly enforce this

Visualizing Training Augmentation

We can take our inputs representations and x:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{x}^T = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}$$

And generate all of the pairs with this matrix:

$$\begin{bmatrix} x_1 \oplus x_1 & x_1 \oplus x_2 & \cdots & x_1 \oplus x_n \\ x_2 \oplus x_1 & x_2 \oplus x_2 & \cdots & x_2 \oplus x_n \\ \vdots & \vdots & \ddots & \vdots \\ x_n \oplus x_1 & x_n \oplus x_2 & \cdots & x_n \oplus x_n \end{bmatrix}$$

Augmentation Impacts

Each part of the matrix enforces a different property by being present during training:

- Diagonal: Self loop [a]
- SUT *or* SLT [b]: general loop consistency
- SUT *and* SLT: Pairwise 'loop'

---

[a] *Very* underrepresented (~1:n)

[b] Strictly Upper/Lower Triangular

$$\begin{bmatrix} x_1 \oplus x_1 & x_1 \oplus x_2 & \cdots & x_1 \oplus x_n \\ x_2 \oplus x_1 & x_2 \oplus x_2 & \cdots & x_2 \oplus x_n \\ \vdots & \vdots & \ddots & \vdots \\ x_n \oplus x_1 & x_n \oplus x_2 & \cdots & x_n \oplus x_n \end{bmatrix}$$

Inference Augmentation

Given a vector of known y values corresponding to the training data, i.e. anchors y^a

$$\mathbf{y^a} = \begin{bmatrix} y_1^a & y_2^a & \cdots & y_n^a \end{bmatrix}^T, \quad \mathbf{y} = \begin{bmatrix} y_1 & y_2 & \cdots & y_m \end{bmatrix}^T$$

We can run inference against all of the anchors using this 'vector product':

$$\mathbf{y^a y}^T = \begin{bmatrix} y_1^a \\ y_2^a \\ \vdots \\ y_n^a \end{bmatrix} \begin{bmatrix} y_1 & y_2 & \cdots & y_m \end{bmatrix} \rightarrow \begin{bmatrix} y_1^a - y_1 & y_1^a - y_2 & \cdots & y_1^a - y_m \\ y_2^a - y_1 & y_2^a - y_2 & \cdots & y_2^a - y_m \\ \vdots & \vdots & \ddots & \vdots \\ y_n^a - y_1 & y_n^a - y_2 & \cdots & y_n^a - y_m \end{bmatrix}$$

But you can also augmented in the *other* direction *if* you have enforced pairwise loop consistency during training, i.e.

$$\mathbf{y^a}^T \mathbf{y} = \begin{bmatrix} y_1^a & y_2^a & \cdots & y_n^a \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \rightarrow \begin{bmatrix} y_1 - y_1^a & y_1 - y_2^a & \cdots & y_1 - y_m^a \\ y_2 - y_1^a & y_2 - y_2^a & \cdots & y_2 - y_m^a \\ \vdots & \vdots & \ddots & \vdots \\ y_n - y_1^a & y_n - y_2^a & \cdots & y_n - y_m^a \end{bmatrix}$$

Mapping back to absolute predictions can be done with averaging.

Augmentation Alternatives

Running inference with this approach is an open research area:

- ML theorists suggest:
  - 'full' augmentation and averaging (Wetzel et al., 2020)
  - various post-training weighting schemes to identify 'best' anchors (Belaid et al., 2024)
  - use K-nearest anchors in embedding space (Netanyahu et al., 2023)
- Chemistry practitioners seem to ignore it:
  - train and infer with one direction (Tynes et al., 2021)
  - check self loop consistency after-the-fact (Fralish et al., 2023)

Initial Investigation

Implementing nepare

The `nepare` package obfuscates the implementation details of the decisions previously discussed:

```
npr = NeuralPairwiseRegressor(
  input_size=2,
  hidden_size=50,
  num_layers=3,
)
```

`NeuralPairwiseRegressor` is a subclass of a
`FeedforwardNeuralNetwork` which is useful for comparison.

Ease of Augmentation

Augmenting training can easily be configured:

```
training_dataset = PairwiseAugmentedDataset(
  X[train_idxs],
  y[train_idxs],
  how='full',  # one of: 'full', 'ut', 'sut'
)
```

Automatic Anchoring

Validation and inference anchoring are also handled behind the scenes:

```
validation_dataset = PairwiseAnchoredDataset(
  X[train_idxs],
  y[train_idxs],
  X[val_idxs],
  y[val_idxs],
  how='full',  # one of: 'full', 'half'
)
predict_dataset = PairwiseInferenceDataset(
  X[train_idxs],
  y[train_idxs],
  X_unknown,
  how='full',  # one of: 'full', 'half'
)
```

Fitting Arbitrary Surfaces

demo notebook

Real Molecular Properties

Small molecule Rat Plasma Protein Binding via `polaris`

- only 110 training samples but high-quality

#	Name	Contributors	pearsonr	
1	1B_MPNN_LargeMix-and-Phenomics		0.908	>
2	1B_MPNN_MolGPS-ens_LargeMix		0.886	>
3	nepare		0.863	>
4	chemma-2b-sft	MB	0.747	>
5	adme-fang-RPPB-1_desc2D_RandomForestRegressor		0.722	>

Figure 3: `nepare` reaches #3 on the leaderboard, narrowly losing to a *billion* parameter foundation model.

Subsequent Steps

Learned Embeddings

Up to this point I have been quietly assuming that some function like this exists:
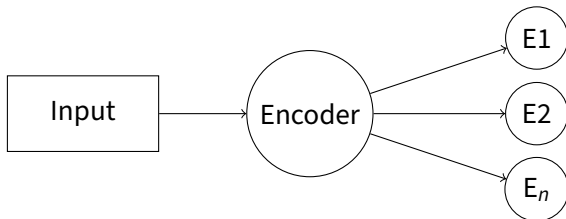
We may want to *learn* this embedding rather than keep it static

- ⁏ Fully differentiable architecture, can readily plug in modules like ChemProp's message passing
- ⁏ DeepDelta has done an incorrect version of this in which they had two separate MP blocks (Fralish et al., 2023)

LearnedEmbeddingNPR

```
class LearnedEmbeddingNPR(NeuralPairwiseRegressor)
    def __init__(self, ..., embedding_module)
```
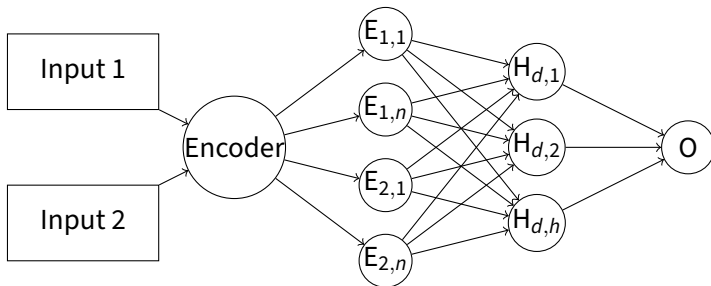
Uncertainty Calibration

I want to provide an in-depth analysis of how well calibrated the model is - need help getting started.

We have some previous work in this group that I am aware of.

I would especially like to know of any existing **frameworks** for doing this so I can avoid re-implementing a bunch of metrics/methods/etc. from scratch.

Acknowledgements

Thank you!

Thanks to all members of the Green Group!

latest group photo (?)

Additional thanks to Nathan Morgan and Jonathan Zheng for their insightful conversations and code contributions.

Supplementary Material

Miscellaneous Thoughts

There are some applications where predicting property *differences* is actually the typical application:

- `confrank`: github.com/grimme-lab/confrank
- all of the Relative Binding Free Energy (RBFE) research space
    - easier to predict difference in BFE than absolute
    - many tools exist to plan 'routes' of low-error difference calculations

Some ML theorists suggest one can enforce pairwise learning without actually changing the target:

- `AdaPRL` train a normal NN but penalize pairwise distances in loss function (https://arxiv.org/abs/2501.05809)

Cited Works

Attia, L., Burns, J. W., Doyle, P. S., & Green, W. H. (2024). Organic solubility prediction at the limit of aleatoric uncertainty. *ChemRxiv*. https://doi.org/10.26434/chemrxiv-2024-93qp3

Belaid, M. K., Wibiral, T., Rabus, M., & Hüllermeier, E. (2024). *Weighted pairwise difference learning*.

Fralish, Z., Chen, A., Skaluba, P., & Reker, D. (2023). DeepDelta: Predicting ADMET improvements of molecular derivatives with deep learning. *Journal of Cheminformatics*, *15*(1). https://doi.org/10.1186/s13321-023-00769-x

Lundberg, S., & Lee, S.-I. (2017). *A unified approach to interpreting model predictions*. https://doi.org/10.48550/ARXIV.1705.07874

Netanyahu, A., Gupta, A., Simchowitz, M., Zhang, K., & Agrawal, P. (2023). *Learning to extrapolate: A transductive approach*. https://arxiv.org/abs/2304.14329

Shen, Z., Yang, H., & Zhang, S. (2022). Optimal approximation rate of ReLU networks in terms of width and depth. *Journal de Mathématiques Pures Et Appliquées*, *157*, 101–135. https://doi.org/10.1016/j.matpur.2021.07.009

Stumpfe, D., Hu, H., & Bajorath, J. (2019). Evolving concept of activity cliffs. *ACS Omega*, *4*(11), 14360–14368. https://doi.org/10.1021/acsomega.9b02221

Tynes, M., Gao, W., Burrill, D. J., Batista, E. R., Perez, D., Yang, P., & Lubbers, N. (2021). Pairwise difference regression: A machine learning meta-algorithm for improved prediction and uncertainty quantification in chemical search. *Journal of Chemical Information and Modeling*, *61*(8), 3846–3857. https://doi.org/10.1021/acs.jcim.1c00670

Wetzel, S. J., Melko, R. G., & Tamblyn, I. (2021). Twin neural network regression is a semi-supervised regression algorithm. *CoRR*, *abs/2106.06124*. https://arxiv.org/abs/2106.06124

Wetzel, S. J., Ryczko, K., Melko, R. G., & Tamblyn, I. (2020). Twin neural network regression. *CoRR*, *abs/2012.14873*. https://arxiv.org/abs/2012.14873

Yousefi, M., & Collins, J. (2024). *Learning the bitter lesson: Empirical evidence from 20 years of CVPR proceedings*. https://arxiv.org/abs/2410.09649