Neural Pairwise Regression

Jackson Burns

7 February 2025

Background

At the most basic level, we want to find some computational model that predicts some target value y based on an input vector x of features (e.g. the value of a house and its square footage and build year, respectively). This can be done with any model architecture - linear regression, classical machine learning algorithms, and neural networks. It has shortcoming in that: - it cannot (and is not intended to) *extrapolate* to new feature *or* target space - predictions are made in a vacuum, i.e. there is no idea of how uncertain we are in a prediction - all models have parameters (θ), and the more complex, expressive models (like NNs, which are universal approximators) that can tackle the hardest problem have a lot of parameters, thus requiring a lot of data to set them

Changing this problem to instead predict the *difference* in a target for two inputs attempts to resolve some of these issues: - extrapolation *can* be easier in this formalism, since Out-of-Combination (OOC) prediction *can* be easier than Out-of-Distribution prediction, see "Learning to Extrapolate" for a theoretical exploration and "Extrapolation is not the same as interpolation" for an emperical example in drug discovery - when running inference, one can use all of the training points as 'anchors' and get a (hopefuly) well calibrated prediction of the uncertainty - we have way more data

Order of questions: Why the interest in this? Low data regimes, some actual delta applications naturally, 'it just works' Why does this work? OOS vs. OOC, more data is more better (or at least, lets you fit a more complex model) What architecture? is technically model agnostic, but NNs are UFAs (important later) How to represent the input? Explicit difference? That's inductive bias a UFA could learn (and could be outright wrong, why would it be manhattan in the feature just because we say so in the target?) How to build the NN? Again, avoid inductive bias in setup, NN can learn it (we have so much data!), i.e. don't need 'branches' or 'operation layers' How do we train? Do we want and/or need to enforce self distance of zero (requires training on self-pairs), loop consistency (free?), and 2-loop consistency (requires training on x12 and x21)? We can attempt to enforce this at the architectural level (satisfy for sure), enforce in loss (satisfy OK), or just check after training (e.g. DeepDelta, probably also OK) How do we run inference? Need to use all points? Subset? of most/least/? similar? Weighting of predictions? e.g. smallest diffs are probably similar to training and thus interpolation, trust them more (just geom mean lol). Or, just take the nice uncertainty estimate and run with it? Choice of training changes what would work here (e.g. could not allow test point to move in the input if we don't have 2-loop consistency) How do we scale this? If this really is just generically more accurate than direct prediction, we certainly want to train everything like this, but now quadratic increase of training data is a burden… how to overcome this? partial augmentation?

```
RBFE informed 'cycle construction' - programs like LOMAP and HIMAP can create a pathway of small perturb
Bayesian Optimization - could select susbsequent anchors to minimize variance
```

Both of these probably lose out to some simple weighting scheme

can we make the triangle inequality, symmetry, cycle etc. be enforeced in delta learning. check what deepdelta dud - also check if we actually need to enforce it? is it just leraning it on its own? i imagine if we put in the self pairs at least that one will be satisfied

There are additional nice things, like how you can use SHAP [1] on this very simple architecture to get an idea of which features for which input are actually contributing to the prediction.

Cited Works

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