Detecting Extrapolation with Influence Functions

David Madras [1 2 3] **James Atwood** [1] **Alex D'Amour** [1]

Abstract

In this work, we explore principled methods for *extrapolation detection*. We define extrapolation as occurring when a model's conclusion at a test point is underdetermined by the training data. Our metrics for detecting extrapolation are based on influence functions, inspired by the intuition that a point requires extrapolation if its inclusion in the training set would significantly change the model's learned parameters. We provide interpretations of our methods in terms of the eigendecomposition of the Hessian. We present experimental evidence that our method is capable of identifying extrapolation to out-of-distribution points.

1. Introduction

As machine learning is deployed in increasingly vital areas, there is increasing demand for metrics that draw attention to potentially unreliable predictions. One important source of unreliability is *extrapolation*. Extrapolation is often used to refer to making predictions made outside of the support of training data. A number of machine learning reliability methods are designed to detect extrapolation in this sense, most notably those aimed at the out of distribution (OOD) detection problem.

While well-defined in theory, the support of the training data can be tricky to define in practice and is, at best, estimable at rates that scale poorly with the dimension of input data. To build metrics that practically measure extrapolation, we focus on a slightly narrower form of the problem. In particular, we say a trained model is *extrapolating* on a test input if the prediction at that test input is *underdetermined* by the training data and model architecture: that is, there exist multiple prediction functions that yield different predictions for the input point, but fit the training/validation data similarly. If a model includes no inductive biases, this notion of extrapolation reduces to the broader notion above. However, when the architecture of the model is restricted, measurements of this narrower notion of extrapolation are practical to estimate, even with high-dimensional inputs, and can indicate "blind spots" in a model's training that may not be apparent from examining the model's internal representations alone.

In this preliminary work, we present some methods for measuring the extent to which a trained model is extrapolating for a particular test input. To do this, we consider various summaries of how the model's learned parameters would change if the test point were included in our training set. We use influence functions to approximate this parameter change. We discuss further intuitions around our method, and how it can be interpreted in terms of eigenvectors of the model's Hessian and sensitivity in parameter space. Experimentally, we show that our method can perform well on OOD tasks, and explore how it reacts differently to models with different inductive biases.

2. Related Work

Some recent works explore the relationship between test points, the learned model, and the training set. Schulam & Saria (2019) use influence functions to assess reliability as measured purely with respect to bootstrap samples of the training set, whereas ours considers points outside of the training distribution. MacKay (1992) presents a related information-theoretic criterion for how much information a test point would add to a model. Several papers examine reliability criteria that are based on distance in some space: within/between-group distances (Jiang et al., 2018), a prespecified kernel in a learned embedding space (Card et al., 2019), or the activation space of a neural network (Papernot & McDaniel, 2018). Our method differs from these in that we explicitly consider which dimensions of the input the model considers (ir)relevant, and do not rely on distance in Euclidean or embedding space as a metric. Additionally, a range of methods exist for more binary (in or out) OOD detection (Choi & Jang, 2018; Liang et al., 2017; Gal & Ghahramani, 2015; Schölkopf et al., 2001) and exploring calibration (Naeini et al., 2015; Guo et al., 2017).

[1]Google Brain, Cambridge, MA, USA [2]University of Toronto, Toronto, ON, Canada [3]Vector Institute, Toronto, ON, Canada. Correspondence to: David Madras <madras@cs.toronto.edu>.

Presented at the ICML 2019 Workshop on Uncertainty and Robustness in Deep Learning.

3. Preliminaries

3.1. Setup

Let $z = (x, y)$ be an example input-output pair, where x is a vector of features and y is a label. We define a model in terms of a loss function $\mathcal{L}$ with parameters θ as a sum over training examples $(z_i)_{i=1}^n$, i.e., $\mathcal{L}(\theta) = \sum_i^N \ell(z_i, \theta)$, where ℓ is an example-wise loss (e.g., 0-1 loss or cross entropy). Let $\theta^\star$ be the parameters of the trained model, obtained by, e.g., minimizing the loss over this dataset, i.e., $\theta^\star = \text{argmin}_\theta \mathcal{L}(\theta)$.

We consider the problem of auditing a trained model in the anomaly detection setting, where unlabeled test points x' arrive one at a time in a stream, and we wish to assess extrapolation on a point-by-point basis. We assume that we can compute gradients $\nabla_\theta \ell(z, \theta^\star)$ for arbitrary examples z.

3.2. Influence Functions

Influence functions are a central tool in our method (Cook & Weisberg, 1982; Koh & Liang, 2017). Originally introduced in robust statistics, influence functions approximate how perturbations to training data translate to changes in a trained model's parameters. Specifically, given a model with trained parameters $\theta^\star$, and an example z', influence functions approximate the change in $\theta^\star$ induced by upweighting z' by some small $\epsilon > 0$ in the loss function $\mathcal{L}$. This upweighting would yield new parameters $\theta^\star_{\epsilon,z'} = \text{argmin}_\theta [\mathcal{L}(\theta) + \epsilon \ell(z', \theta)]$. Cook & Weisberg (1982) define this influence as

$$\mathcal{I}(z') = \frac{d\theta^\star_{\epsilon,z'}}{d\epsilon}\Big|_{\epsilon=0} = -H^{-1}_{\theta^\star} \nabla_{\theta^\star} \ell(z', \theta^\star). \quad (1)$$

where H is the Hessian of the model. This method essentially takes a Newton step in a quadratic bowl formed around $\theta^\star$. In many ML applications, the loss function is non-convex, even locally around the trained parameters $\theta^\star$ (Sagun et al., 2017), and so we make use of a regularized Hessian $H_{\theta^\star,\lambda} = H_{\theta^\star} + \lambda I$, where λ is chosen so that $H_{\theta^\star,\lambda}$ is positive definite, as suggested by Koh & Liang (2017). We denote the influence computed with this regularized Hessian as $\mathcal{I}_\lambda(z')$. We discuss methods for computing these quantities using only gradient information in Section 4.3.

4. Method

In this section, we describe a set of methods for measuring extrapolation on a test input. Our method is built off of an intuition about a duality between extrapolation at prediction time and information at training time. Specifically, if the prediction at a test input x' is underdetermined by the training data, then a training example at that input would be highly informative, and thus have high influence on the model's parameters and predictions. Each heuristic that we propose here measures some aspect of this influence.

4.1. Measuring influence of test points

Here, we provide a general specification for our extrapolation heuristics. Our heuristics operate at prediction time on unlabeled test inputs x', but estimate the potential influence of labeled training instances with input values x'. To do this, for each test input x', we construct a set of candidate training examples $\{(x', y')\}_{y' \in \mathcal{Y}}$, where $\mathcal{Y}$ is a set of candidate labels. We then aggregate our metrics over the full set $\mathcal{Y}$, where the aggregation can be a function like $\min, \max$, mean or variance. Currently, the set of candidate labels and the aggregation function are left as tuning parameters. To make decisions using the aggregated influence value, we compare it to the distribution of the same aggregated values calculated on a held-out validation set.

We define several extrapolation metrics using this general specification. Our first metric simply measures the squared norm of the influence on model parameters.

$$\mathcal{P}_\lambda(x') = \text{agg}_{y' \in \mathcal{Y}} \|\mathcal{I}_\lambda((x', y'))\|_2^2. \quad (2)$$

Our second metric measures the influence that an example at the test input x' would have on its own prediction, inspired by notions of leverage in the linear regression setting (Chatterjee et al., 1986). Following the formulation in Koh & Liang (2017),

$$\mathcal{A}_\lambda(x') = \text{agg}_{y' \in \mathcal{Y}} \nabla_\theta \ell((x', y'), \theta^\star)^\top \mathcal{I}_\lambda((x', y')). \quad (3)$$

Our final metric is based on the recognition that the influence $I_\lambda((x', y'))$ is sensitive to the choice of λ, but that this sensitivity is indicative of indeterminacy in the prediction at x'. Thus, we directly measure this sensitivity at z' by evaluating the squared norm of the derivative with respect to λ:

$$\mathcal{D}_\lambda(x') = \text{agg}_{y' \in \mathcal{Y}} \left\| \frac{d\mathcal{I}_\lambda((x', y'))}{d\lambda} \right\|_2^2, \quad (4)$$

where

$$\begin{aligned} \frac{d\mathcal{I}_\lambda(z)}{d\lambda} &= -\frac{dH^{-1}_{\theta^\star,\lambda}}{d\lambda} \nabla_\theta \ell(z, \theta^\star) \\ &\overset{(\star)}{=} -H^{-1}_{\theta^\star,\lambda} \frac{dH_{\theta^\star,\lambda}}{d\lambda} H^{-1}_{\theta^\star,\lambda} \nabla_\theta \ell(z, \theta^\star) \\ &= -H^{-2}_{\theta^\star,\lambda} \nabla_\theta \ell(z, \theta^\star), \end{aligned}$$

and where $(\star)$ uses the identity for a derivative of a matrix inverse (Petersen et al., 2008). We note that, using this same identity, $\frac{d\mathcal{A}_\lambda(x)}{d\lambda} = \mathcal{P}_\lambda(x)$.

4.2. Interpretation of Metrics

Although each of these metrics has a slightly different motivation, they are each aggregations of a quadratic form

involving the gradient of the loss at the candidate test point and powers of the inverse Hessian:

$$\nabla_\theta \ell(z, \theta^\star)^\top H_{\theta^\star,\lambda}^{-k} \nabla_\theta \ell(z, \theta^\star). \tag{5}$$

In particular, setting $k = 1$ yields $\mathcal{A}_\lambda$; setting $k = 2$ yields $\mathcal{P}_\lambda$; and setting $k = 4$ yields $\mathcal{D}_\lambda$.

Casting our metrics in this way allows an interpretation in terms of the eigendecomposition of the regularized Hessian, similar to the interpretation provided in Schulam & Saria (2019). The Hessian $H_{\theta^\star,\lambda}$, which we assume has all eigenvalues $\lambda \neq 0$, can be decomposed into UDU^T, where U contains the eigenvectors of $H_{\theta^\star,\lambda}$, $(\xi_{(1)}, \cdots, \xi_{(p)})$, and D is a diagonal matrix with eigenvalues, $(d_{(1)}, \cdots, d_{(p)})$, on the diagonal. Using this decomposition, we rewrite the quadratic form in (5) as

$$\sum_{j=1}^{p} d_{(j)}^{-k} (\nabla_\theta \ell(z, \theta^\star)^\top \xi_{(j)})^2.$$

For a candidate example z', this quadratic form has large norm if the candidate gradient $\nabla_\theta \ell(z', \theta^\star)$ is aligned with eigenvectors of $H_{\theta^\star,\lambda}$ with small eigenvalues, corresponding to directions in the parameter space θ in which the loss $L(\theta)$ has low curvature around the chosen parameters $\theta^\star$. The loss evaluated on the training data provides little information to discriminate between parameter values in these directions. We expect that test examples whose gradients align with these dimensions to have predictions that are underdetermined by the training data.

4.3. Computational Issues

Computing influence functions for complex models is not trivial. The first challenge is inverting the Hessian. For large models with thousands or millions of parameters, even instantiating the Hessian is expensive, let alone inverting it. To get around this, we use tricks for calculating Hessian-vector products implicitly (HVPs). These methods allow one to calculate the product Hv for any vector v – where H is the implicit Hessian of a function f, for which we only have oracle access to $f(u)$ and $\nabla f(u)$ (Pearlmutter, 1994).

Using these HVP methods, Koh & Liang (2017) outline two ways to calculate the product of the *inverse* Hessian with a vector (iHVP). The first involves transforming the iHVP problem into an optimization problem: $H^{-1}v = \text{argmin}_u \frac{1}{2} u^T H u - u^T v$, assuming that H is positive definite. Then this optimization problem is solved using conjugate gradient (CG) descent. The second method involves using the second-order *LiSSA* optimization method from Agarwal et al. (2017) to approximate the Taylor expansion of H^{-1} — this method is intended to be more robust to stochastic (minibatch) estimates of H.

5. Experiments

In this section, we show experimental evidence that our methods can detect extrapolation in some simple models. First, we present experiments with toy data to gain intuition. Then, we apply our techniques to OOD detection for convolutional neural networks (CNNs) trained on MNIST. We note that while these two tasks do not quite test the subtle extrapolation detection we are ultimately interested in, standard OOD detection work usually examines more extreme distribution shift than the tasks we present (e.g. train on CIFAR, test on SUN — see (Hendrycks & Gimpel, 2016; Liang et al., 2017) for examples). Therefore, we think these tasks are useful for our purposes; the OOD data "looks like" the training data, but still requires extrapolation.

5.1. Toy Data

We generate 1-D data points $y = \beta x^2 + \mathcal{N}(0, 1)$, where $\mathcal{N}$ represents a sample from a Gaussian distribution and $\beta \in \mathbb{R}$. Our training set consists of x drawn uniformly from $[-0.5, 0.5]$ and $[2.5, 3.5]$. However, at test time, we will consider $x \in [-3, 6]$. We expect stronger evidence of extrapolation for extreme values of $x > 3.5$ or $x < -0.5$, some mild extrapolation between the modes of the training distribution for $x \in [0.5, 2.5]$, and no extrapolation when the test input overlaps with the training distribution.

We generate 200 training points and train three small neural networks (2 hidden layers of 5 units each) with different activation functions on this data. We show results here only for the ReLU network (Figure 2); see Appendix A for results from Tanh and Softplus networks, demonstrating these models' varying inductive biases. Next, we calculate $\mathcal{A}_\lambda(x), \mathcal{P}_\lambda(x)$ and $D_\lambda(x)$ for a number of test inputs $x \in [-6, 9]$, with y varying across the output range of the training data. This creates a grid of test points which range from high to zero probability under the training distribution.

In Figure 1, we show $\mathcal{A}, \mathcal{P}, \mathcal{D}$ using $\max$ aggregation for the ReLU network over various values of x, λ. Note that we are able to calculate and invert the Hessian exactly for these models, due to their smaller number of parameters (~ 50). In general, we observe the desired W-shaped behaviour for all three metrics — the extreme-valued inputs have higher scores, the in distribution have lowest, and the inputs between the training modes have slightly higher values. We note some interesting phenomena. Firstly, the metric values general increase as λ decreases. This is as expected — since the value of λ is effectively the value of the smallest (positive) eigenvalue of H, we would expect the inverse of H to grow proportional to $\frac{1}{\lambda}$. Interestingly, the desired W-shape holds for values of λ which are smaller in magnitude than the smallest eigenvalue of H, meaning that the influence function is meaningful even when $H + \lambda I$ is not positive definite. Secondly, the results for the higher powers of H

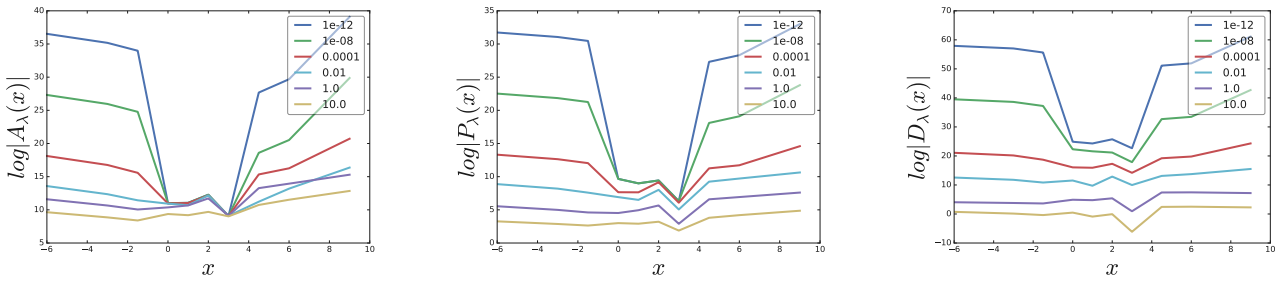


Figure 1. From left to right: $\mathcal{A}_\lambda(x), \mathcal{P}_\lambda(x), \mathcal{D}_\lambda(x)$ for a range of λ with a small two-layer neural network with ReLU hidden units. In-distribution x are at 0 and 3. Aggregation function is $\max$.

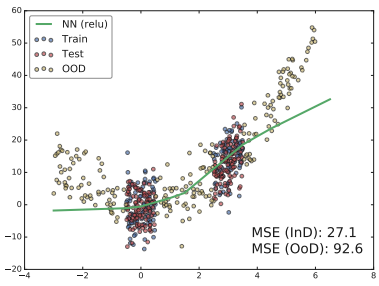


Figure 2. Data and trained ReLU model. See Appendix A for other models. In-distribution points $x \in [-\frac{1}{2}, \frac{1}{2}] \cup [2\frac{1}{2}, 3\frac{1}{2}]]$ are blue and red; OOD points are yellow.

Metric	Noise $= 0$
$\mathcal{A}_\lambda(x)$	0.920
$\mathcal{P}_\lambda(x)$	0.919
$\mathcal{D}_\lambda(x)$	0.920
Max Prediction	0.916
NN-Activations	**0.960**
NN-Pixels	0.819
Ens: Activs+Preds	**0.968**
Ens: Activs+$\mathcal{P}_\lambda(x)$	**0.970**

Table 1. MNIST OOD Experiment for $\lambda = 3$. AUC for OOD prediction shown. Aggregation function is weighted average. Max Prediction baseline uses the largest predicted probability. NN-Activations uses 1-NN in concatenated activation space. NN-Pixels uses 1-NN in pixel space. "Ens" denotes ensembling of methods by p-value averaging. Best methods within 95%-CI in bold.

($\mathcal{P}$ and $\mathcal{D}$) are more consistent in shape across values of λ. We note that when using implicit Hessian techniques, these small values of λ are not feasible since we need the Hessian to be positive definite. We noted qualitatively similar results in the tanh and softplus networks — see Appendix B.

5.2. MNIST

To explore our method's ability to scale up to larger datasets and models, we present experiments using a CNN on the handwritten digit dataset MNIST. The CNN we trained has two convolutional layers and one fully connected layer, containing ~ 8000 parameters. We separated the 5s as an OOD class and trained on the other 9 digits. Then we test by calculating each metric for a batch of validation data as a calibration set, as well as batches of held-out in-distribution and OOD data. For each held-out point, we can calculate a p-value (Shafer & Vovk, 2008): the percentage of validation points which have more extreme values of that metric. We then report the AUC of this p-value for OOD detection.

We used CG descent (see Sec. 4.3) to calculate the IHVP. We used 1000 points from each of the validation, test, and OOD sets. We used a weighted average for the aggregation function — the value of each $\mathcal{I}_\lambda((x, y'))$ was weighted by the predicted probability of the label at that point by the network (see Appendix C). This can be framed as an approximation to the Bayesian evidence criterion proposed in MacKay (1992). In Table 2, we show our metrics ($\mathcal{A}, \mathcal{P}$, and $\mathcal{D}$) outperform some standard baselines, and can be useful when ensembled with others. We note that the strongest baseline metric is taking the distance to the nearest neighbour in the validation set in activation space across the network (Papernot & McDaniel, 2018). These results suggest there is promise in our proposed class of methods for detecting extrapolation in trained models.

6. Conclusion

In this work, we presented a method for detecting extrapolation in a trained model. We use influence functions as a major building block of this method, and discuss the relationships between this method and other previously proposed approaches from robust machine learning. In future work, we hope to scale up these methods to larger models and datasets, and to test on tasks more suited to what we believe are the unique strengths of our method. We also hope to explore applications in fairness and interpretability, where connecting test points to training points is of paramount importance.

Acknowledgements

Thanks to Jamie Smith, Yaniv Ovadia, Yoni Halpern, and Pang Wei Koh for helpful comments and discussions.

References

Agarwal, N., Bullins, B., and Hazan, E. Second-order stochastic optimization for machine learning in linear time. *The Journal of Machine Learning Research*, 18(1): 4148–4187, 2017.

Card, D., Zhang, M., and Smith, N. A. Deep weighted averaging classifiers. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, pp. 369–378. ACM, 2019.

Chatterjee, S., Hadi, A. S., et al. Influential observations, high leverage points, and outliers in linear regression. *Statistical science*, 1(3):379–393, 1986.

Choi, H. and Jang, E. Generative ensembles for robust anomaly detection. *arXiv preprint arXiv:1810.01392*, 2018.

Cook, R. D. and Weisberg, S. *Residuals and influence in regression*. New York: Chapman and Hall, 1982.

Gal, Y. and Ghahramani, Z. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. arxiv, 2015.

Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1321–1330. JMLR. org, 2017.

Hendrycks, D. and Gimpel, K. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136*, 2016.

Jiang, H., Kim, B., Guan, M., and Gupta, M. To trust or not to trust a classifier. In *Advances in Neural Information Processing Systems*, pp. 5541–5552, 2018.

Koh, P. W. and Liang, P. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1885–1894. JMLR. org, 2017.

Liang, S., Li, Y., and Srikant, R. Enhancing the reliability of out-of-distribution image detection in neural networks. *arXiv preprint arXiv:1706.02690*, 2017.

MacKay, D. J. The evidence framework applied to classification networks. *Neural computation*, 4(5):720–736, 1992.

Naeini, M. P., Cooper, G., and Hauskrecht, M. Obtaining well calibrated probabilities using bayesian binning. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

Papernot, N. and McDaniel, P. Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning. *arXiv preprint arXiv:1803.04765*, 2018.

Pearlmutter, B. A. Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160, 1994.

Petersen, K. B., Pedersen, M. S., et al. The matrix cookbook. *Technical University of Denmark*, 7(15):510, 2008.

Sagun, L., Evci, U., Guney, V. U., Dauphin, Y., and Bottou, L. Empirical analysis of the hessian of over-parametrized neural networks. *arXiv preprint arXiv:1706.04454*, 2017.

Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J., and Williamson, R. C. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7): 1443–1471, 2001.

Schulam, P. and Saria, S. Auditing pointwise reliability subsequent to training. *arXiv preprint arXiv:1901.00403*, 2019.

Shafer, G. and Vovk, V. A tutorial on conformal prediction. *Journal of Machine Learning Research*, 9(Mar):371–421, 2008.

Metric	WtdAvg	Min	Max	Var	Mean
$\mathcal{A}_\lambda(x)$	0.919	0.867	0.454	0.432	0.327
$\mathcal{P}_\lambda(x)$	0.919	0.864	0.478	0.452	0.348
$\mathcal{D}_\lambda(x)$	0.921	0.873	0.511	0.485	0.382
Max Pred	0.897				

Table 2. AUCs for MNIST OOD Experiment for $\lambda = 1$ and 10% label noise. Max Prediction baseline uses the largest predicted probability.

A. Trained Neural Networks on Toy Data

In Figure 3, we show the results of the trained models on toy data. Note that the only difference was the activation function. The models result in similar predictions inside the training domain but differ outside of it, indicating extrapolation.

B. Influence Functions on Toy Data

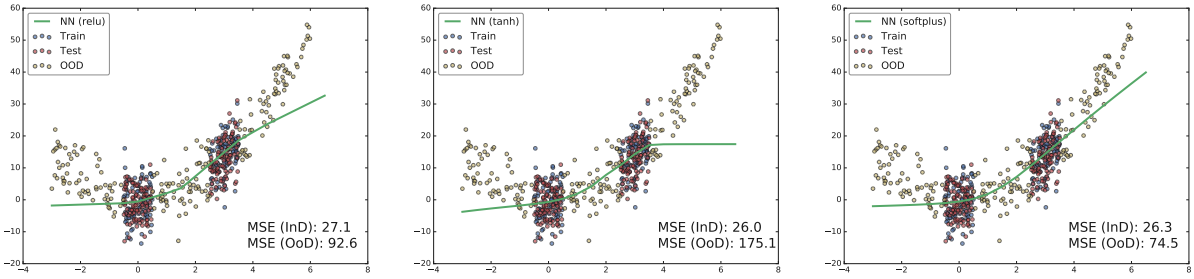
In Figures 4, 5, and 6, we show the results for each network (ReLU, Tanh, Softplus) for $\mathcal{A}_\lambda(x), \mathcal{P}_\lambda(x)$, and $\mathcal{D}_\lambda(x)$ respectively.

C. Aggregation Functions

Here, we show that WtdAvg is the only aggregation function which beats the baseline. $\min$ also does almost as well, but $\max$, mean, and variance all perform at or worse than random chance. Further work is needed to understand the properties of these aggregation functions, and when each is suitable - we note that in the 1-D experiment, $\max$ and variance seemed to outperform $\min$.

D. Batch Effects of CG Optimization

When solving the IHVP problem with CG, we estimated the HVP for minibatches of points, to improve speed. We noticed some batch effects — different minibatches of CG computations had different statistics. We hypothesize that even though we do not share parameters between examples, sharing a tolerance parameter between examples caused the resolution of the optimization of each example to depend on the other examples in the batch. We were not able to run CG for all points at once due to memory restrictions, so this is a potential concern — it is possible that different batches of CG will give wildly different results, and therefore p-values calculating by comparing results across CG batches are invalid. In Figures 7 and 8, we show boxplots for each metric across each of 50 CG batches (each consisting of 20 validation points) for MNIST with 0 and 10% label noise respectively. We notice batch effects more strongly in the lower noise version. To counteract the batch effects, we subtracted the median validation value for that minibatch from each metric in order to compare between batches. This yielded a performance improvement, particularly for the stronger batch effects in the noiseless version.



(a) ReLU network (b) Tanh network (c) Softplus network

Figure 3. Three small neural networks trained on a one-dimensional regression problem. Note that each network has similar error in-distribution (InD) but very different error out-of-distribution (OOD). This differing behaviour outside the training domain reflects underdetermination of that problem in that area of the input space — hence the networks must extrapolate (i.e., rely on inductive biases) for those inputs.

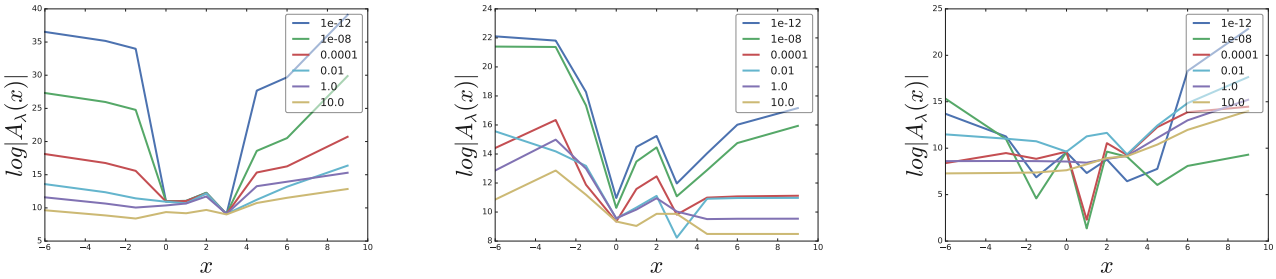


Figure 4. $|\mathcal{A}_\lambda(x)|$ for a range of λ on 1-D data. In-distribution x are at 0 and 3. Aggregation function is $\max$.

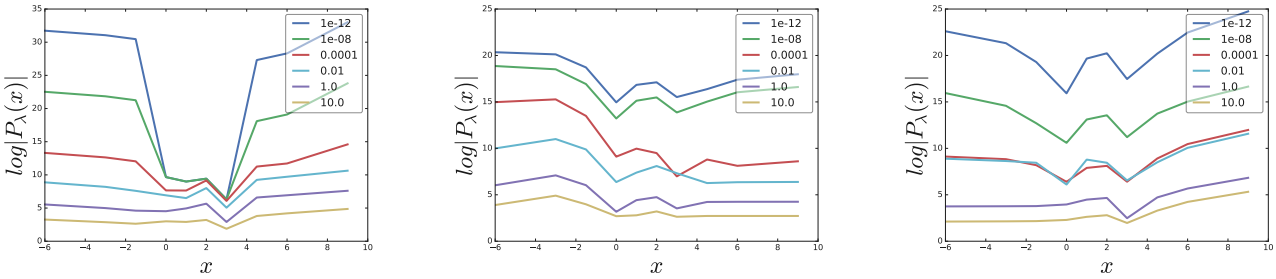


Figure 5. $|\mathcal{P}_\lambda(x)|$ for a range of λ on 1-D data. In-distribution x are at 0 and 3. Aggregation function is $\max$.

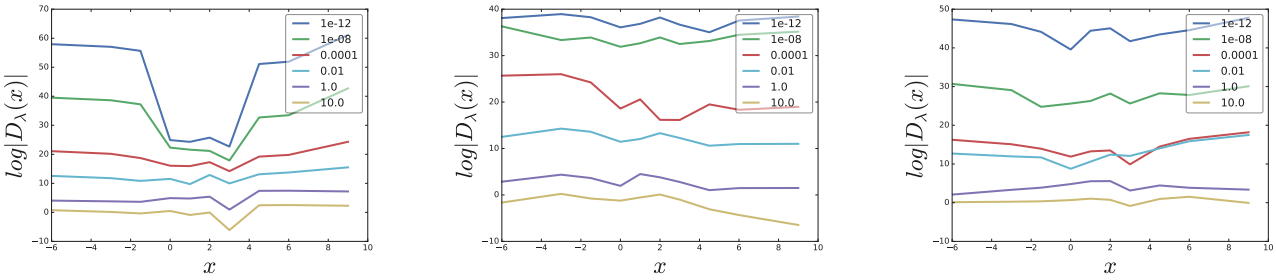


Figure 6. $|\mathcal{D}_\lambda(x)|$ for a range of λ on 1-D data. In-distribution x are at 0 and 3. Aggregation function is $\max$.

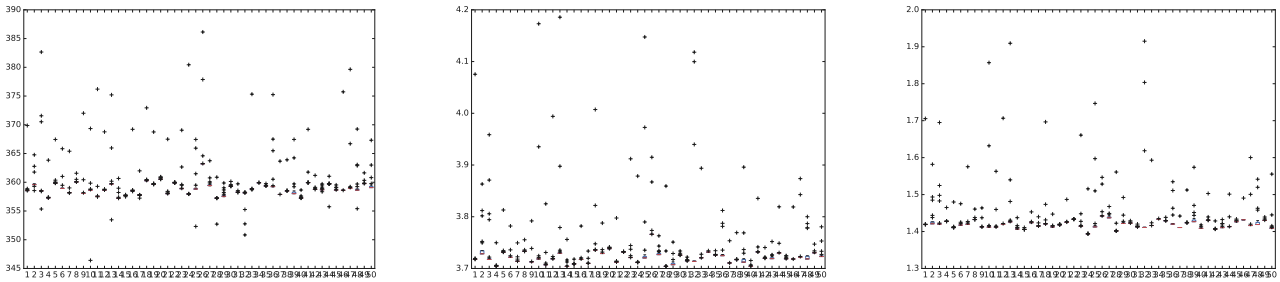


Figure 7. Batch effects for $|\mathcal{A}_\lambda(x)|, |\mathcal{P}_\lambda(x)|, |\mathcal{D}_\lambda(x)|$ on MNIST. Each batch contains 20 validation points.

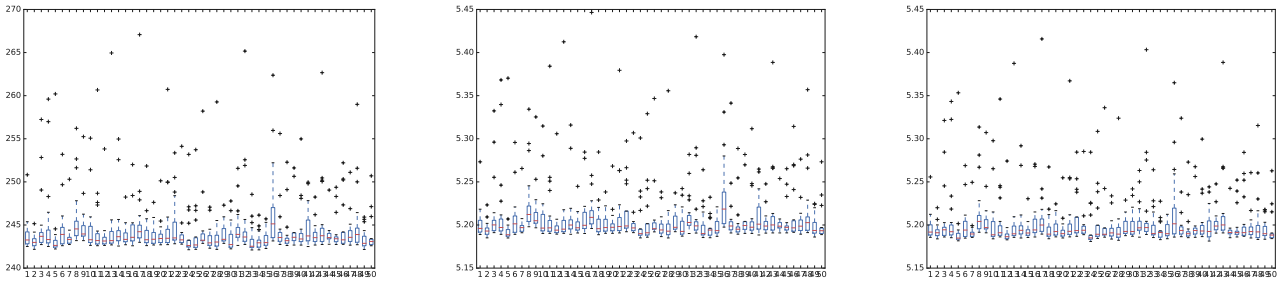


Figure 8. Batch effects for $|\mathcal{A}_\lambda(x)|, |\mathcal{P}_\lambda(x)|, |\mathcal{D}_\lambda(x)|$ on MNIST with 10% noise. Each batch contains 20 validation points.