Deep Black-Box Optimization with Influence Functions

Jayanth Koushik[1] Michael J. Tarr[1] Aarti Singh[1]

Abstract

Deep neural networks are increasingly being used to model black-box functions. Examples include modeling brain response to stimuli, material properties under given synthesis conditions, and digital art. In these applications, often the model is a surrogate and the goal is rather to optimize the black-box function to achieve the desired brain response, material property, or digital art characteristics. Moreover, resource constraints imply that, rather than training on a passive dataset, one should focus subsequent sampling on the most informative data points. In the Bayesian setting, this can be achieved by utilizing the ability of Bayesian models such as Gaussian processes to model uncertainty in observed data via posterior variance, which can guide subsequent sampling. However, uncertainty estimates for deep neural networks are largely lacking or are very expensive to compute. For example, bootstrap or cross-validation estimates require re-training the network several times which is often computationally prohibitive. In this work, we use influence functions to estimate the variance of neural network outputs, and design a black-box optimization algorithm similar to confidence bound-based Bayesian algorithms. We demonstrate the effectiveness of our method through experiments on synthetic and real-world optimization problems.

1 Introduction

Black-box optimization, also known as zeroth order optimization, is the problem of finding the global minima or maxima of a function given access to only (possibly noisy) evaluations of the function. Perhaps the most popular black-box optimization approach is in the Bayesian setting, such as Gaussian process (GP) optimization, which assumes that the black-box function is sampled from a GP, and uses an acquisition function such as lower confidence bound (LCB) to guide sampling and subsequently update the posterior mean and variance of the GP model in an iterative manner. However, recently, deep neural networks are increasingly being used to model black-box functions. Examples include modeling brain

[1]Carnegie Mellon University. Email: jayanthkoushik@cmu.edu, michaeltarr@cmu.edu, aartisingh@cmu.edu.

response to stimuli[1–3], material properties under given synthesis conditions[4], and digital art[5]. Often the goal in these problems is optimization of the black-box model, rather than learning the entire model. For example, a human vision researcher might be interested in understanding which images cause maximum activation in a specific brain region[6,7]; a material scientist is interested in finding optimal experimental conditions that yield a material with desired properties[4] or generate digital art with desired characteristics[5]. While a simple approach is to learn a deep model on passively acquired evaluations of the function, and then report its optima, this is wasteful as often the black-box evaluations are expensive (c.f., subject time in a brain scanner is limited, material synthesis experiments are expensive, etc.). Also, often pre-trained models of black-boxes need to be updated subsequently to identify inputs that may lead to novel outputs not explored in training set. For example, in material science a model trained to predict the energy of a pure lattice may need to be updated to understand new low-energy configurations achievable under defects, or deep neural net models of images may need to be updated to achieve desired characteristics of synthetic digital images. Thus, it is of interest to develop sequential optimization methods akin to Bayesian optimization for deep neural networks.

Sequential optimization of neural network models requires an acquisition function, similar to Bayesian optimization. However, popular acquisition functions (LCB, expectation maximization, Thompson sampling, etc.) are mostly based on an uncertainty measure or confidence bound which characterizes the variability of the predictions. Unfortunately, formal methods for uncertainty quantification that are also computationally feasible for deep neural network models are largely non-existent. For example, bootstrap or cross-validation based estimates of uncertainty require re-training the network several times, which is typically computationally prohibitive. In this paper, we seek to investigate principled and computationally efficient estimators of uncertainty measures (like variance of prediction), that can then be used to guide subsequent sampling for optimization of black-box functions.

Specifically, we use the concept of influence functions[8,9] from classical statistics to approximate the leave-one-out cross-validation estimate of the prediction variance, *without having to re-train the model*. It is known[9] that if the loss function is twice-differentiable and strictly convex, then the influence function has a closed-form approximation, and the influence-function based estimate provides an asymptotic approximation of the variance of prediction. Even though the loss function of neural networks is non-differentiable and non-convex, it was recently shown[10] that in practice, the approximation continues to hold for this case. However, Koh and Liang (2017)[10] used influence functions to understand the importance of each input on the prediction of a passively trained deep neural

network, Influence functions were not investigated for uncertainty quantification and estimation of prediction variance for use in subsequent sampling.

A related line of work is activation maximization in neural networks (NNs) where the goal is to find input that maximizes the output of a particular unit in the network. However, since the corresponding target functions are known and differentiable, gradient based optimization methods can be used. Still, obtaining results suitable for visualization requires careful tuning and optimization hacks [11]. In this paper, we will consider the activation maximization problem in a black-box setting, to mimic neuroscience and material science experiments, where the brain is the black-box function. Furthermore, prior work is passive requiring learning a good model for all inputs, while we focus on collecting new data to sequentially guide the model towards identifying the input which leads to maximum output *without necessarily learning a good model for all inputs*.

There have also been attempts to directly extend Bayesian optimization to neural networks. Snoek et al. (2015) [12] add a Bayesian linear layer to neural networks, treating the network outputs as basis function. Springenberg et al. (2016) [13] focus on scalability, and use a Monte Carlo approach, combined with scale adaptation. However, our focus is to enable sequential optimization of existing NN models being used in scientific domains. Our contributions can be summarized as follows:

- We use influence functions to obtain a computationally efficient approximation of prediction variance in neural networks.
- We propose a computationally efficient method to compute influence functions for neural network predictions. Our approach uses a low-rank approximation of the Hessian, which is represented using an auxillary network, and trained along with the main network.
- We develop a deep black-box optimization method using these influence function based uncertainty estimates that is valid in the non-Bayesian setting.
- We demonstrate the efficacy of our method on synthetic and real datasets. Our method can be comparable, and also outperform Bayesian optimization in settings where neural networks may be able to model the underlying function better than GPs.

The rest of the paper is organized as follows. In Section 2, we formally define the problem, and the Bayesian setting we build upon in this work. Our proposed method is described in Section 3, followed by results on synthetic and real datasets in Section 4. We conclude with discussion of open problems in Section 5.

2 Preliminaries

2.1 Problem Setting

We consider the problem of sequential optimization of a black-box function. Specifically, let $f : \mathcal{X} \to \mathbb{R}$ be a cost function to be minimized. At each step t, we select a point $x_t \in \mathcal{X}$, and observe a noisy evaluation $y_t = f(x_t) + \epsilon_t$, where ϵ_t is independent o-mean noise. This noisy evaluation is the only way to interact with the function, and we don't assume any prior knowledge of it. We will use z to denote an input-output pair; $z \equiv (x, y) \in \mathcal{X} \times \mathbb{R}$.

Practical functions in this category (like hyper-parameter optimization for instance) generally tend to be "expensive", either in terms of time, or resources, or both. This makes it impractical to do a dense "grid search" to identify the minimum; algorithms must use as few evaluations as possible. With a given time budget T, the objective is to minimize the simple regret, $\min_{t=1\ldots T} f(x_t) - f(x^*)$ where $x^* \in \arg\min_{x\in\mathcal{X}} f(x)$ is a global minimum (not necessarily unique). This measures how close to the optimum an algorithm gets in T steps, and is equivalent to minimizing $\min_{t=1\ldots T} f(x_t)$.

2.2 Bayesian Optimization

Bayesian optimization is a popular method for solving black-box optimization problems, which uses GP models to estimate the unknown cost function. At each step T, newly obtained data (x_T, y_T) is used to update a GP prior, and the posterior distribution is used to define an acquisition function $\alpha_T : \mathcal{X} \to \mathbb{R}$. The next point to query, x_{T+1} is selected by minimizing the acquisition function; $x_{T+1} = \arg\min_{x\in\mathcal{X}} \alpha_T(x)$. Popular acquisition functions are expected improvement (EI), maximum probability of improvement (MPI), and LCB. Here, we will particularly focus on LCB, which provides the motivation for our method.

2.3 GP-LCB

Consider a GP model with mean function 0, and covariance function $k(\cdot, \cdot)$. After observing T points, the model is updated to obtain a posterior mean function μ_T, and a posterior covariance function $k_T(\cdot, \cdot)$. The LCB acquisition function is $\alpha_T^{LCB}(x) = \mu_T(x) - \beta_T^{1/2}\sigma_T(x)$, where $\sigma_T(x) \equiv k_T(x, x)$, and β_T is a parameter for balancing exploration and exploitation. This expression is easily interpretable; μ_T is an estimate of expected cost, and σ_T is an estimate of uncertainty. The next point is chosen taking both into consideration. Points with low expected cost are exploited, and points with high uncertainty are explored. α_T defines a "pessimistic" estimate (lower confidence bound) for the cost, hence the name of the algorithm.

3 Method

Suppose we have a neural network $g : \mathcal{X} \to \mathbb{R}$ with parameters $\theta \in \Theta$, trained using a convex loss function $L : (\mathcal{X} \times \mathbb{R}) \times \Theta \to \mathbb{R}^+$. At a particular step T, we get an estimate of the parameters $\hat{\theta}_T$, by minimizing $(1/T)\sum_{t=1}^{T} L(z_t, \theta)$. As noted earlier, $z_t \equiv (x_t, y_t)$, and we will use g_θ to denote the network with a particular set of parameters. Now, for any point $x \in \mathcal{X}$, we have a prediction for the cost function, i.e., $g_{\hat{\theta}_T}(x)$. So, if we get an estimate $\hat{\sigma}_T(x)$, for the variance of $g_{\hat{\theta}_T}(x)$, we can define an acquisition function $\alpha_T(x) = g_{\hat{\theta}_T}(x) - \hat{\sigma}_T(x)$. Then optimization proceeds similar to Bayesian optimization; where we select the next point for querying x_{T+1} by minimizing $\alpha_T(X)$. This auxiliary optimization problem is non-convex, but local minima can be obtained through gradient based methods. In practice, it is also common to use multiple restarts when solving this problem, and select the best solution. We now describe our method for estimating the variance using influence functions.

3.1 Influence Functions

Intuitively, influence functions measure the effect of a small perturbation at a data point on the parameters of a model. We up-weight a particular point z^+ from the training set $\{z_t\}_{t=1}^{T}$, and obtain a new set of parameters $\hat{\theta}_T^+(z^+, \nu)$ by minimizing the reweighted loss function, $(1/T)\sum_{t=1}^{T} L(z_t, \theta) + \nu L(z^+, \theta)$. We define the influence of z^+ on $\hat{\theta}_T$ as the change $\hat{\theta}_T^+(z^+, \nu) - \hat{\theta}_T$ caused by an adding an infinitesimal weight to z^+. Formally,

$$\mathcal{I}_{\hat{\theta}_T}\Big(z^+\Big) = \lim_{\nu \to 0} \frac{\hat{\theta}_T^+(z^+, \nu) - \hat{\theta}_T}{\nu} = \frac{\partial \hat{\theta}_T^+(z^+, \nu)}{\partial \nu}.$$

Importantly, the influence function can be approximated using the following result.

$$\mathcal{I}_{\hat{\theta}_T}\Big(z^+\Big) \approx -H_{\hat{\theta}_T}^{-1} \nabla_\theta L\Big(z^+, \hat{\theta}_T\Big),$$

where $\nabla_\theta L(z^+, \hat{\theta}_T)$ is the gradient of the loss with respected to the parameters evaluated at $(z^+, \hat{\theta}_T)$, and $H_{\hat{\theta}_T} \equiv (1/T)\sum_{t=1}^{T} \nabla_\theta^2 L(z_t, \hat{\theta}_T)$ is the Hessian. Now, we can use the chain rule to extend this approximation to the influence on the prediction of $g_{\hat{\theta}_T}$. For a test point $x^\dagger$, let $\mathcal{I}_{g_{\hat{\theta}_T}}\Big(x^\dagger, z^+\Big)$ be the influence of z^+ on $g_{\hat{\theta}_T}\Big(x^\dagger\Big)$. So,

$$
\begin{aligned}
\mathcal{I}_{g_{\hat{\theta}_T}}\left(x^\dagger, z^+\right) &= \frac{\partial g_{\hat{\theta}^+_T(z^+,\nu)}\left(x^\dagger\right)}{\partial \nu} \\
&= \frac{\partial g_{\hat{\theta}_T}\left(x^\dagger\right)}{\partial \theta} \frac{\partial \hat{\theta}^+_T(z^+,\nu)}{\partial \nu} \\
&= \frac{\partial g_{\hat{\theta}_T}\left(x^\dagger\right)}{\partial \theta} \mathcal{I}_{\hat{\theta}_T}\left(z^+\right) \\
&\approx -\frac{\partial g_{\hat{\theta}_T}\left(x^\dagger\right)}{\partial \theta} H^{-1}_{\hat{\theta}_T} \nabla_\theta L\left(z^+, \hat{\theta}_T\right).
\end{aligned}
$$

3.2 Variance Estimation

Finally, we estimate the variance by computing the average squared influence over the training points.

$$
\hat{\sigma}_T(x) = \frac{1}{T} \sum_{t=1}^{T} \mathcal{I}_{g_{\hat{\theta}_T}}(x, z_t)^2.
$$

In semi-parametric theory, influence is formalized through the behavior of asymptotically linear estimators, and under regularity conditions, it can be shown that the average squared influence converges to the asymptotic variance[14].

3.3 Implementation

The procedure described above cannot be directly applied to neural networks since the Hessian is not positive-definite in the general case. We address this issue by making a low-rank approximation, $H_{\hat{\theta}} \approx Q \equiv PP^T$. Let $P = U\Sigma V^T$ be a singular value decomposition (SVD) of P. Then, $Q^\dagger \equiv U\Sigma^{\dagger^2}U^T$ is the Moore-Penrose pseudoinverse of Q, where $\Sigma^{\dagger^2}$ is a diagonal matrix with reciprocals of the squared non-zero singular values. With this, for any vector v we can approximate the product with the inverse Hessian.

$$
H^\dagger_\theta v \approx Q^\dagger v = U\Sigma^{\dagger^2}U^T v.
$$

We represent the low-rank approximation using a second neural network with a single hidden layer. The network uses shared weights similar to an autoencoder, and given input v, computes PP^Tv. We train this network to approximate $H_{\hat{\theta}}\nabla_\theta L(z, \hat{\theta})$, using samples from the training data. The Hessian vector product can be computed efficiently by performing two backward passes through the network (Perlmutter's method). After updating the network at each step T, the SVD of P is computed, which allows efficient computation of $H^{-1}_{\hat{\theta}_T}\nabla_\theta L(z^+, \hat{\theta}_T)$. The full algorithm (NN-INF) is shown in Figure 1.

hyper-parameter $\{\beta_t\}_{t=1\ldots\infty}$ ▷ exploration-exploitation trade-off values
hyper-parameter n_p ▷ random samples used to pre-train the model
hyper-parameter r ▷ Hessian approximation rank
hyper-parameter n_H ▷ samples used for training Hessian approximation
hyper-parameter n_I ▷ samples used for computing influence

procedure NNINF(f, $\mathcal{X}$, T, g_θ)
▶ Minimize f over $\mathcal{X}$ for T steps using the network g_θ.

$D \leftarrow \{(x, f(x)) : x \in \textsc{Sample}(\mathcal{X}, n_p)\}$ ▷ samples for pre-training
$|\theta| \leftarrow$ number of parameters in θ
$P \leftarrow \textsc{Matrix}(|\theta|, r)$ ▷ for low rank Hessian approximation

for $t \leftarrow 1 \ldots T$ **do**
$\textsc{TrainNetwork}(g_\theta, D)$
$P, \mathcal{I} \leftarrow \text{IHVP}(g_\theta, D, P)$
$x_t \leftarrow \arg\min_{x \in \mathcal{X}} \textsc{Acquisition}(x, g_\theta, \mathcal{I}, \beta_t)$
$D \leftarrow D \cup \{(x_t, f(x_t))\}$
end for

return $\arg\min_{(x,y)\in D} y$
end procedure

procedure IHVP(g_θ, D, P)
▶ Compute $H_\theta^{-1} \nabla_\theta L(z, \theta)$ for $z \in D$.

$\pi_P \leftarrow \textsc{FullyConnectedNetwork}\big(P, P^T\big)$
$L_H \leftarrow \{\nabla_\theta L(z, \theta) : z \in \textsc{Sample}(D, n_H)\}$
$J_\theta \leftarrow (1/n_H) \sum_{v \in L_H} v$
$\nu_J \leftarrow \nabla_\theta J_\theta$
$D_H \leftarrow \{\big(v, \nabla_\theta v^T \nu_J\big) : v \in L_H\}$

$\textsc{TrainNetwork}(\pi_P, D_H)$
$U, \Sigma, V \leftarrow \text{SVD}(P)$
$W \leftarrow U \Sigma^{\dagger^2}$

$\mathcal{I} \leftarrow \{W U^T v : v \in \textsc{Sample}(L_H, n_I)\}$
return P, $\mathcal{I}$
end procedure

procedure $\textsc{Acquisition}(x, g_\theta, \mathcal{I}, \beta)$
▶ compute the acquisition function at x

$\mu \leftarrow g_\theta(x)$
$\nu_\mu \leftarrow \nabla_\theta \mu$
$\sigma \leftarrow \sqrt{\frac{1}{n_I} \sum_{\iota \in \mathcal{I}} (\mathbf{1}^T \nu_\mu \iota)^2}$
return $\mu - \beta^{1/2} \sigma$
end procedure

Figure 1: Algorithm

3.4 GP-INF

The influence approximation for variance can also be applied to GP models, by viewing them as performing kernel ridge regression. In this case, there is a closed form expression for the influence [15], so we can directly compute variance approximation. This gives a method similar to GP-LCB, where we use the influence approximation of variance instead of the posterior variance. We term this method GP-INF, and use it as an additional baseline in our experiments.

4 Experiments

4.1 Synthetic function maximization

First, we compare our method with GP based algorithms using common test functions used in optimization: five dimensional Ackley function [16], and ten dimensional Rastrigin function [17]. For the Ackley function, we use a network with 3 hidden layers, with 8, 8, and 4 hidden units respectively. And for the Rastrigin function, we again use a network with 3 hidden layers, but with 16, 16, and 8 hidden units. In both cases, we approximate the Hessian with a rank 5 matrix. We report two sets of results, using different schemes for setting the β_t parameter (used in INF and LCB methods).

Figure 2 (1) shows the instantaneous regret over 500 iterations with $\beta_t = c\sqrt{t}\log^2(10t)$ (based on the theoretical results presented by Srinivas et al. (2009) [18]). We set $c = 0.1$ for GP methods, and $c = 0.01$ for NN-INF. We did not find c to have a significant effect on performance, but for consistency, we used scaled β_t for NN-INF by 10 in all cases. Figure 2 (2) shows the same results, but with β_t held constant throughout the experiment. We have $\beta_t = 2$ for GP methods, and $\beta_t = 0.2$ for NN-INF.

4.2 Neural network output maximization

We now demonstrate results on a task inspired from neuroscience. To understand the properties of a neuron, brain region etc., experimenters collect response signals (neuron firing rate, increase in blood oxygenation etc.) to different stimuli in order to identify maximally activating inputs. This is generally done in an ad-hoc manner, whereby experimenters hand pick, or manually create a restricted set of images designed to address a given theoretical question. This can lead to biased results caused by insufficient exploration of the input space. One way to address this issue is to perform adaptive stimulus selection over the full input space.

To simulate the setting of stimulus selection in neuroscience, we first trained a convolutional neural network (CNN) to classify images from the MNIST dataset. The output layer of this CNN has 10 units, each corresponding to one

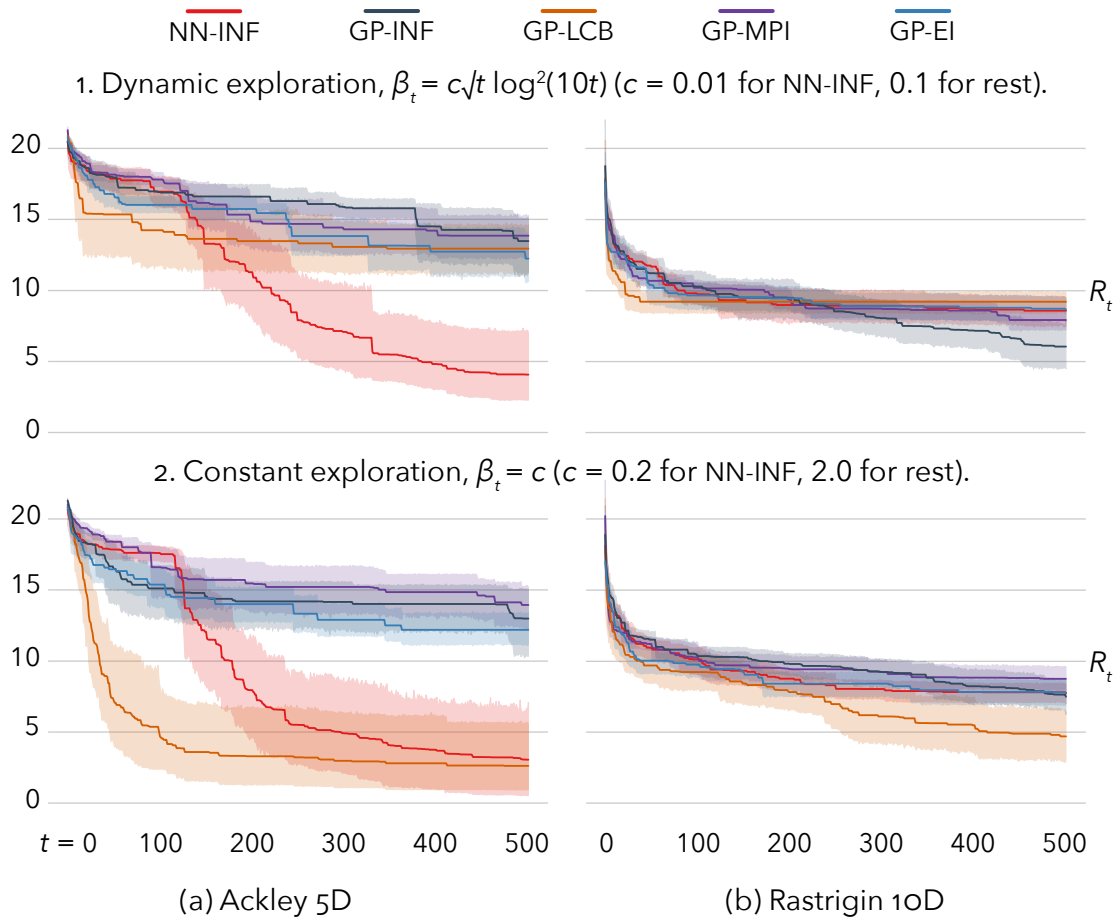

Figure 2: Optimization of synthetic functions

of the MNIST digits (0 to 9). Given an input image, the output of each unit is proportional to the probability (as predicted by the model), that the image belongs to the particular class. With this, we can define an optimization task: find the image that maximizes the output of a particular unit. This is similar to a neuroscience visual stimulus selection experiment, where the output unit could be a single neuron in the visual cortex.

Given the difficultly of this optimization problem, it is important to exploit available prior knowledge. For a visual experiment, this could be in the form of a pre-trained network. Here, we pre-train our CNN model for binary classification of two digits different from the target digit; for example (classifying '5' vs. '6' when the target digit is '2'. For the model, we use a smaller CNN than the target; with two convolution layers, each with a single filter. Figure 3 shows the target neuron output for two different settings. In Figure 3 (a), the target digit is '2', and the CNN is pre-trained for classifying '5' vs. '6'. In Figure 3 (b), the

target digit is '3', and the CNN is pre-trained for classifying '1' vs. '8'. In both cases, we see that the CNN model is able to exploit the prior information, and achieve better performance compared to the GP-LCB baseline. This is a promising result showing the feasibility of large scale adaptive sitmulus selection.

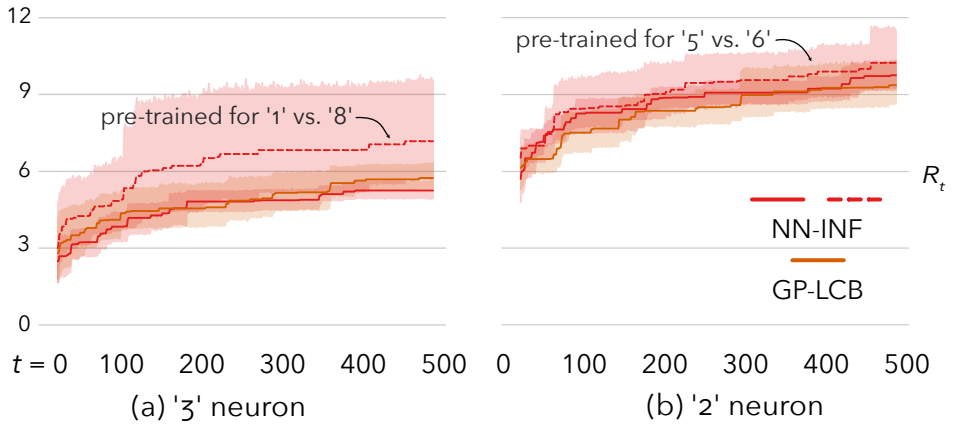

Figure 3: MNIST

5 Discussion

In this paper, we use the notion of influence functions from classical statistics to estimate the variance of prediction made using a neural network model, without having to retrain the model on multiple subsets of data as in bootstrap or cross-validation based estimates. We additionally use these uncertainty estimates to design a deep black-box optimization algorithm, that can be used to optimize a black-box function such as brain response or desired material property with sequentially collected data. We show the efficacy of our algorithm on synthetic and real datasets.

There are several directions for future work. First, the uncertainty estimates we propose are backed by theoretical underpinning under convexity assumptions when the samples are assumed to be independent and it is of interest to develop theoretical guarantees for the non-convex and sequentially dependent samples setting which arises in optimization. The latter should be possible given parallel analysis in the Bayesian setting. Such non-Bayesian confidence bounds that are valid for sequential data can then also be used for active learning of black-box functions or deep models. Second, while the method does not require retraining the NN model at each iteration for variance estimation, the model does require retraining as new data is collected. While this is inevitable

in optimization and active learning settings,the computational complexity can be improved by not training the model to convergence at each iteration. For example, in [19], and references therein, computational efficiency is achieved for active learning of linear separators by training the model to lower accuracy initially (e.g. it should be matched to the lower statistical accuracy due to limited samples initially) and then increasing the computational accuracy at subsequent iterations. Finally, we have only explored the notion of uncertainty (coupled with prediction maximization) to guide subsequent sampling. However, since neural networks learn a feature representation, another way to guide sampling is via the notion of expressiveness (c.f. [20]) that selects data points which help improve the learnt feature representation. It is interesting to compare and potentially combine the notions of uncertainty and expressiveness to guide sampling for optimization as well as active learning of black-box functions modeled via deep neural networks.

References

[1] Daniel LK Yamins, Ha Hong, Charles F Cadieu, Ethan A Solomon, Darren Seibert, and James J DiCarlo. Performance-optimized hierarchical models predict neural responses in higher visual cortex. *Proceedings of the National Academy of Sciences*, 111(23):8619–8624, 2014.

[2] Pulkit Agrawal, Dustin Stansbury, Jitendra Malik, and Jack L Gallant. Pixels to voxels: Modeling visual representation in the human brain. *arXiv preprint arXiv:1407.5104*, 2014.

[3] Alexander JE Kell, Daniel LK Yamins, Erica N Shook, Sam V Norman-Haignere, and Josh H McDermott. A task-optimized neural network replicates human auditory behavior, predicts brain responses, and reveals a cortical processing hierarchy. *Neuron*, 98(3):630–644, 2018.

[4] D. Xue, P. V. Balachandran, J. Hogden, J. Theiler, and T. Lookman. Accelerated search for materials with targeted properties by adaptive design. *Nature communications*, 7:11241, 2016.

[5] Lev Manovich. Data science and digital art history. *International Journal for Digital Art History*, (1), 2015.

[6] Carlos R Ponce, Will Xiao, Peter Schade, Till S Hartmann, Gabriel Kreiman, and Margaret S Livingstone. Evolving super stimuli for real neurons using deep generative networks. *bioRxiv*, page 516484, 2019.

[7] Pouya Bashivan, Kohitij Kar, and James J DiCarlo. Neural population control via deep image synthesis. *Science*, 364(6439):eaav9436, 2019.

[8] R Dennis Cook and Sanford Weisberg. Characterizations of an empirical influence function for detecting influential cases in regression. *Technometrics*, 22(4):495–508, 1980.

[9] R Dennis Cook and Sanford Weisberg. *Residuals and influence in regression*. New York: Chapman and Hall, 1982.

[10] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1885–1894. JMLR. org, 2017.

[11] Anh Nguyen, Alexey Dosovitskiy, Jason Yosinski, Thomas Brox, and Jeff Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. In *Advances in Neural Information Processing Systems*, pages 3387–3395, 2016.

[12] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *International conference on machine learning*, pages 2171–2180, 2015.

[13] Jost Tobias Springenberg, Aaron Klein, Stefan Falkner, and Frank Hutter. Bayesian optimization with robust bayesian neural networks. In *Advances in Neural Information Processing Systems*, pages 4134–4142, 2016.

[14] Anastasios Tsiatis. *Semiparametric theory and missing data*. Springer Science & Business Media, 2007.

[15] Viktoria Öllerer, Christophe Croux, and Andreas Alfons. The influence function of penalized regression estimators. *Statistics*, 49(4):741–765, 2015.

[16] David Ackley. *A connectionist machine for genetic hillclimbing*, volume 28. Springer Science & Business Media, 2012.

[17] LA Rastrigin. Systems of extremal control. *Nauka*, 1974.

[18] Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*, 2009.

[19] Pranjal Awasthi, Maria Florina Balcan, and Philip M. Long. The power of localization for efficiently learning linear separators with noise. *J. ACM*, 63(6):50:1–50:27, January 2017. ISSN 0004-5411. doi: 10.1145/3006384. URL `http://doi.acm.org/10.1145/3006384`.

[20] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489*, 2017.