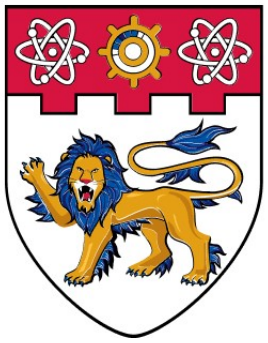

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Training-Free Neural Active Learning with Initialization Robustness Guarantees

Singh Jasraj

School of Physical and Mathematical Sciences
Nanyang Technological University

In partial fulfilment of the requirements for the degree of
Bachelor of Science (Honors)

November 6, 2024

Supervisor: Prof. Low Kian Hsiang Bryan
Co-Supervisor: Prof. Tong Ping

Acknowledgements

I wish to convey my profound appreciation to my supervisor, Professor Low Kian Hsiang Bryan, and my collaborator, Hemachandra Apivich, for their steadfast backing, direction, and motivation during my research adventure. Their knowledge, forbearance, and astute advice have been instrumental in molding my dissertation and fostering my scholarly development.

Additionally, I sincerely thank my co-supervisor, Professor Tong Ping, for his valuable comments, helpful critiques, and generous support. His proficiency and contributions have been essential in improving the caliber and influence of my investigation.

I am immensely grateful to my parents, Kaur Charanjeet and Singh Sarabjit, and my sister, Kaur Gursimran, for their boundless affection and continual encouragement. Their sacrifices and efforts have been the cornerstone of my academic journey, and I am indebted to them for all they've done.

Finally, I want to convey my gratitude to my friends – Agarwal Gopal, Anand Arukshita, Gupta Chirag, Panwar Samay, Tan Hui Ru Kimberley, and Tayal Aks – for their unwavering support, encouragement, and motivation. Their friendship and companionship kept me grounded, motivated, and sane throughout my thesis.

I reiterate my heartfelt thanks to everyone who has provided assistance and motivation during my scholarly endeavors.

Abstract

Neural active learning techniques so far have focused on enhancing the predictive capabilities of the networks. However, safety-critical applications necessitate not only good predictive performance but also robustness to randomness in the model-fitting process. To address this, we present the *Expected Variance with Gaussian Processes* (EV-GP) criterion for neural active learning, which is theoretically guaranteed to choose data points that result in neural networks exhibiting both (a) good generalization capabilities and (b) robustness to initialization. Notably, our EV-GP criterion is training-free, i.e., it does not require network training during data selection, making it computationally efficient. We empirically prove that our EV-GP criterion strongly correlates with initialization robustness and generalization performance. Additionally, we demonstrate that it consistently surpasses baseline methods in achieving both objectives, particularly in cases with limited initially labeled data or large batch sizes for active learning.

Contents

List of Figures

Chapter 1

Introduction

1.1 Problem Motivation

Neural network training is known to be sensitive to the initialization of the network [12, 36]. These networks are usually randomly initialized [18, 22] and then trained according to some iterative gradient-based method. Consequently, the network function at convergence will be dependent on initialization as will be the predictions it makes.

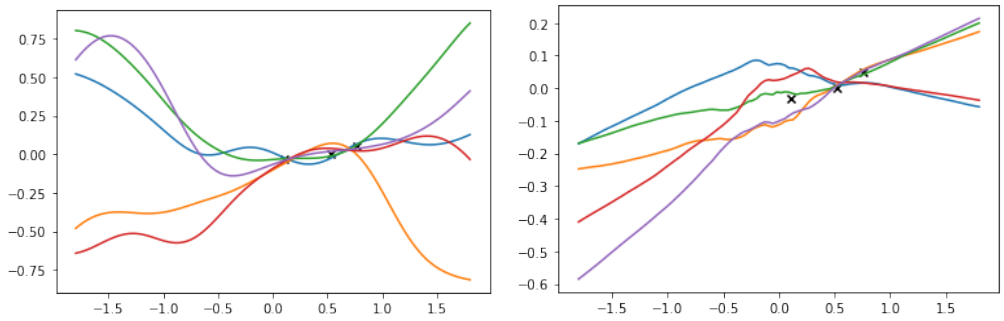

Figure 1.1: Demonstration of the effect of random initialization on the converged network. Five randomly initialized 5-layer neural networks with widths 1024 trained using gradient descent on 3 data points. *Left:* networks with Radial Basis Function (RBF) activation. *Right:* networks with Rectified Linear Unit (ReLU) activation.

Figure 1.1 shows the converged functions obtained by training multiple neural nets with different initializations. The predictions have high variability in some regions of the input space, which lowers the model's reliability since we cannot be sure that the trained network will give us consistent results. Moreover, we have to hope that the initialization of the network is favorable for our task. There are practical reasons for ensuring that the model training is initialization-robust:

1. **Reliability** – We can trust the trained network more if we can ensure that the predictions made by it do not vary too much due to randomness in its initialization; it is important to ensure that it will give answers which are not subject to chance.

2. **Explainability** – When choosing specific data points for training, it's essential to establish a connection between the training process using the queried data and the performance of the resulting model. Moreover, when the model generates a prediction, we aim to quantify the extent of uncertainty attributable to the inherent randomness within the model.

Simply retraining the network several times and selecting the run which resulted in the best performance (according to some metric) can help control the error due to randomness in network initialization. This is feasible in cases where the cost of data acquisition is low, and the computation budget is high. However, this method cannot be used in low-resource settings.

Since network convergence depends on its initialization, it can be interesting to explore ways to lower this dependence. One aspect that could be controlled is the choice of the dataset used for training the network. Data quality might interest a party when the cost of data procurement is high. This may not be a concern for large companies with enough resources to acquire large volumes of data. However, it can be valuable for smaller businesses with limited resources. The cost of acquiring training data may not even be monetary. In the healthcare setting, for example, labeling is done by experts, and the data needs to be anonymized, which incurs a high cost that adds up if this needs to be done for the whole dataset. In cases where no party holds a monopoly on the data but is instead distributed amongst multiple parties, there may be communication costs to ask for data from each party.

In cases where only a subset of data is to be labeled, it is hard to determine the effect of the selected points on the performance of the trained network. Therefore, the best strategy would be to select points that can guarantee good performance of the converged network, regardless of the (currently unknown) model initialization. With a neural network model, additional considerations are required:

1. Since neural networks are expensive to train, the active learning algorithm should ideally require no model fitting to select the points to query labels for. Moreover, it is more convenient for the concerned party to collect all the points at once instead of in batches.

2. We should be able to link the subset chosen directly with how uncertain the predictions will be. Ideally, the algorithm should select data points that minimize the uncertainty in the predictions due to random initialization. Moreover, the algorithm should also be able to guarantee the generalizability of the trained model.

3. The algorithm should require little to no modifications to the training workflow. In other words, there should not be a need to change the neural network architecture or the training process (into, say, a Bayesian neural network, which can more explicitly capture the predictive uncertainty) to accommodate the active learning algorithm.

To this end, several works have proposed techniques for performing *neural active learning* – selecting a small subset of data points to be labeled and be used for training a neural

network (see Subsection 2.4.4). However, these methods have primarily focused on improving generalization performance but have not considered robustness metrics. Consequently, in our work, we consider the variant of neural active learning problem which addresses the issues of *initialization-robustness* while adhering to the abovementioned concerns.

1.2 Contributions

In this work, we introduce the *Expected Variance with Gaussian Processes* (EV-GP) criterion (Section 3.4) which selects data points that simultaneously lead to low generalization error and high initialization robustness. For this, we use the characterization of training dynamics given by Neural Tangent Kernel [26] (Section 2.4), and use the predictive distribution of the trained network to measure the output variance. We also introduce an approximation of the output variance and prove a theoretical bound on its quality. Finally, we prove that this approximation provides an upper bound for the generalization error. Hence, our criterion

1. is label-independent and therefore, does not need the heuristic of pseudo-labels,
2. is training-free, i.e., its calculation does not require any network training and is, hence, able to sidestep significant computational expenses,
3. only requires calculating the variance at individual test points rather than the full covariance over the testing set,
4. can make use of the approximation techniques based on sparse Gaussian processes (Subsection 2.3.2) for which we replace the full-rank posterior with its sparse counterparts,
5. minimizes the output variance at individual test points, as well as bounds the generalization error at them,
6. is monotone submodular, and therefore a greedy approach involving selecting the points which give the largest increase in the criterion is guaranteed to provide a $(1 - 1/e)$-optimal solution.

We also present the results of extensive regression and classification experiments to demonstrate that our EV-GP criterion performs better than existing baselines in terms of both predictive performance and initialization robustness.

Chapter 2

Background

In this chapter, we review the relevant literature for our work. First, in Section 2.2 we present the current progress in neural active learning and identify the gap that our proposed work aims to fill. Then, in Section 2.3 we discuss Gaussian processes and sparse posterior approximations for computation speed-ups. Finally, in Section 2.4 we discuss the neural tangent kernel and its relevance to our work.

2.1 Problem Setting

Consistent with previous works on active learning (AL), we assume that we have an unlabeled pool of data $\mathbf{X}_U \subseteq \mathcal{X}$ which we can query labels for, where $\mathcal{X}$ is the input space. Also suppose that we have some test points $\mathbf{X}_T \subseteq \mathcal{X}$ which we want our network to eventually make predictions on. Note that in our case we do not assume that $\mathbf{X}_T$ and $\mathbf{X}_U$ are the same, or that $\mathbf{X}_T$ is a subset of $\mathbf{X}_U$. Also note that we do not require the true labels $\mathbf{y}_T$ of the test data for our AL problem.

We assume to start with the labeled set $\mathcal{L}_0 = (\mathbf{X}_0, \mathbf{y}_0)$, which might be empty. Our algorithm should proceed to select a subset of data points, $\mathbf{X}_L \subset \mathbf{X}_U$, with $|\mathbf{X}_L| \leq k$, where k is the budget for our algorithm – a limit on how many labels can be queried from the oracle. In the pool-based AL setting, we are able to see all of $\mathbf{X}_U$ in each round whereas in streaming setting of AL, unlabeled data arrives in a stream and the algorithm must choose to query the arriving data point or to ignore it [8]. In each round, the algorithm will select b unlabeled points, $\mathbf{X}_b \subset \mathbf{X}_U$, to submit to the oracle, where b is called the batch size. For each data point in a batch, $\mathbf{x}_i \in \mathbf{X}_b$, the oracle will return a noisy observation, $y_i = f(\mathbf{x}_i) + \xi_i$, where f is the true underlying function and ξ_i is i.i.d. noise. We will collectively refer to the observations for the queried points as $\mathbf{y}_L$.

Once the budget has been exhausted, the queried data $\mathcal{D} = (\mathbf{X}_L, \mathbf{y}_L)$ can be used to train the neural network $f(\cdot; \theta)$. The parameters are initialized using the Xavier initialization [18] (see Subsection 2.4.1 for more details), $\theta_0 \sim \text{init}(\theta)$, and the network is trained using the

regularized Mean-Squared Error (MSE) loss:

$$\mathcal{L}\left(\mathcal{D};\theta\right) = \frac{1}{|\mathcal{D}|}\sum_{(\mathbf{x},y)\in\mathcal{D}}\frac{1}{2}\|f\left(x;\theta\right)-y\|^2 + \frac{\lambda}{2}\|\theta\|^2$$

where λ controls the trade-off between minimizing the MSE term and the regularization term. For our our theoretical analysis, we restrict ourselves to the regularized MSE loss. In practice however, other loss functions such as the cross-entropy loss may be used. We assume that the network is trained until convergence to obtain the parameters $\theta_\infty = \text{train}\left(\theta_0\right)$, where $\text{train}\left(\cdot\right)$ is the network training function. As discussed in Chapter 1, our aim is to ensure that the converged model has low generalization error and the predictions have low variance with respect to variability in θ_0.

2.2 Neural Active Learning

AL is a well-studied problem within classical machine learning [59]. In the case of neural networks, however, the problem is less straight-forward due to a number of reasons:

1. An AL algorithm for neural networks should be able to select points to query in batches. This is because the network performance is not noticeably affected by the addition of one point in the training pool. Hence, selecting one point at a time is inefficient.

2. The AL algorithm must not require retraining of the network every time some points are labeled since neural networks are very expensive to train.

3. The randomness in the network may affect its output and hence, should be taken into account. It may arise from multiple sources such as model initialization and stochastic gradients (if one is to train the network using stochastic optimization algorithms [56]). This variability may affect the test loss, measured accuracy or measured uncertainty.

4. It is difficult to reason about a neural network's uncertainty due to a lack of a probabilistic interpretation. There are methods to interpret the output of neural networks as probabilities, for example, by treating the softmax output as class probabilities [7]. However, they may not be a true reflection of the uncertainty of the model [49]. There are also Bayesian neural networks which can more explicitly capture the predictive uncertainty [38, 44]. However, this may require alterations in the training procedure.

Despite these limitations, AL with neural networks has been extensively studied, and its application can be seen in areas such as computer vision, natural language processing, and more [55]. The currently proposed algorithms can mainly be placed on a spectrum – on one end are algorithms based on sample diversity, and on the other are those based on uncertainty in the model prediction.

2.2.1 Diversity-Based Algorithms

In diversity-based AL, the goal is to select a subset of data based on how similar the data points are according to some representation. These measures can be thought of as functions which indicate how similar (or dissimilar) two samples are. A good subset of data, therefore, should be *diverse*, so that it is a good representation of the underlying input space.

Discriminative Models on the Input Distribution

One way to select a diverse set of data points for labeling is to train a discriminative model that can be used to determine if points similar to a given unlabeled point have already been included in the active set. For example, [17] proposes a method which uses a binary classifier to directly discriminate between the set of labeled points and that of unlabeled points. Then, the algorithm can select the points which the discriminator predicts to be different from the labeled set. Upon receiving the labels for these points from the oracle, the discriminator can be updated with the new labeled set and the process repeats.

Instead of discriminating the input data directly, we can do so instead on some latent space. For instance, the Variational Adversarial Active Learning (VAAL) algorithm uses a variational autoencoder (VAE) for learning an embedding function that can be used for discriminating the labeled set from the unlabeled set [62].

Certain improvements have also been made to VAAL to incorporate information in the network function evaluation at the data points. For instance, [73] proposed state-relabeling VAAL, which uses the predictive uncertainty along with the latent representations to discriminate between data points. In a similar vain, [28] proposed task-aware VAAL, which instead uses the predictive loss to aid the discriminator.

A limitation of these methods is that they require an additional model, usually another neural network, to learn from the samples which have already been selected by the algorithm. This adds to the computation cost of selecting the points since we have to train this addition model on the labeled points. Moreover, we need to engineer a discriminator that can distinguish between the latent representations, and the errors form these sources accumulate.

Distribution Matching

An alternative method for choosing the active set involves treating the problem as a distribution matching problem. This entails creating an active set that closely approximates a target distribution, typically the distribution of the unlabeled data. Several measures of distribution divergence have been proposed in AL research:

- Maximum Mean Discrepancy (MMD) – The MMD between two distributions p and q on $\mathcal{X}$ is defined as

$$\mathrm{MMD}\left(p, q, \mathcal{F}\right) = \sup_{f \in \mathcal{F}} \left| \mathbb{E}_{\mathbf{x} \sim p}\left[f\left(\mathbf{x}\right)\right] - \mathbb{E}_{\mathbf{x}' \sim q}\left[f\left(\mathbf{x}'\right)\right] \right|$$

where $\mathcal{F}$ describes a class of functions $f : \mathcal{X} \to \mathbb{R}$. $\mathcal{F}$ is often defined to be a unit ball in some Reproducing Kernel Hilbert Space (RKHS), in which case $\text{MMD}(p, q, \mathcal{F}) = 0$ if and only if $p = q$ [19]. If the MMD is non-zero, it can indicate the degree of dissimilarity between the distributions. Moreover, the MMD can be calculated empirically by utilizing the sample mean as an approximation for the expected value over the distribution. To use the MMD as a measure of distribution divergence, the problem can be reformulated as a linear optimization problem whose objective is to minimize the empirical MMD between the active set and the currently unlabeled samples [9].

- $\mathcal{H}$-Divergence – The $\mathcal{H}$-divergence between two distributions p and q on $\mathcal{X}$ is defined as

$$d_{\mathcal{H}}(p, q) = 2 \sup_{h \in \mathcal{H}} \left| \mathbb{P}_{\mathbf{x} \sim p(\mathbf{x})} (h(\mathbf{x}) = 1) - \mathbb{P}_{\mathbf{x} \sim q(\mathbf{x})} (h(\mathbf{x}) = 1) \right|$$

  where $\mathcal{H}$ is the hypothesis class [5]. The $\mathcal{H}$-divergence between the active set and the unlabeled set can be minimized by training a classifier that can be trained to distinguish between these sets of points [17].

- Wasserstein Distance – The Wasserstein m-distance between two distributions p and q on $\mathcal{X}$ is defined as

$$W_m(p, q) = \left(\inf_{\gamma \in \Gamma(p,q)} \mathbb{E}_{(x,y) \sim \gamma} d(x, y)^m \right)^{1/m}$$

  where $\Gamma(p, q)$ is the set of all probability measures on $\mathcal{X} \times \mathcal{X}$ such that

$$\int \gamma(x, y)\, dy = p(x) \quad \text{and}$$
$$\int \gamma(x, y)\, dx = q(y)$$

  Intuitively, the Wasserstein m-distance measures the minimum cost required to morph the probability distribution p into q, and hence, draws similarity with the edit distance. The AL process proposed by [61] involves alternating between selecting a new batch of data that minimizes the Wasserstein 1-distance and finding the optimal hypothesis through model training. However, computing the exact Wasserstein 1-distance is NP-hard and challenging to approximate. To address this issue, a technique based on Kantorovich-Rubinstein duality can be used to reformulate the computation of the Wasserstein 1-distance as a min-max optimization problem that involves training additional models.

Output Diversity

Another approach for selecting an active set involves using a embedding function, h, and performing diversification on the latent space. The latent space is expected to have a reliable similarity metric, d, which can be used by existing diversification algorithms to choose

a high-quality core set.

A good representation for the input data could just be the model prediction, i.e., $h(x) = f(x;\theta)$ [58]. With this representation, the AL problem can be rephrased as a k-center problem, where the goal is to construct an active set that minimizes the backward Hausdorff distance to $\mathbf{X}_T$:

$$\mathbf{X}_L = \underset{\substack{\mathbf{X}\subset\mathbf{X}_U \\ |\mathbf{X}|\leq k}}{\operatorname{argmin}} \max_{y\in\mathbf{X}_T} \min_{x\in\mathbf{X}} d\left(h\left(x\right), h\left(y\right)\right)$$

We can then use either a greedy algorithm to solve the problem. An integer linear program can be solved to obtain a better solution.

Discussion

In this section, we have reviewed various methods for selecting data points based on measures of diversity. We have observed that common approaches rely on using a metric to measure similarity (or dissimilarity) between the data points or their latent representations. One major limitation of diversity-based methods for AL is that they typically rely on constructing a representation of the data to achieve diversification. This often involves training an additional, often costly, model, or otherwise using some heuristic that may not have a direct connection with the predictive ability of the neural network. In such a case, the algorithm may not provide strong guarantees about the predictive uncertainty of the network trained on the queried data.

2.2.2 Uncertainty-Based Algorithms

In uncertainty-based AL, the goal is to select points to query based on the prediction made by some model. An input can be judged based on the degree of uncertainty in predicting its label, or on how correlated the prediction is to that on another input. In machine learning, there are two main types of uncertainty - aleatoric and epistemic. Aleatoric uncertainty, or sometimes referred to as data uncertainty, is due to the inherent randomness in the true labels. An example of data with high aleatoric uncertainty would be points which lie close to the decision boundary. Meanwhile, epistemic uncertainty, which is sometimes referred to as model uncertainty, is due to *incomplete knowledge*. Regions where enough training data is available will have low epistemic uncertainty, whereas regions where few points lie will have high epistemic uncertainty. Epistemic uncertainty can be reduced if the data selection algorithm chooses to query more points from under-represented regions. In this literature review, we will concentrate on how to utilize these uncertainty measures to construct active learning criteria.

Model Prediction as Degree of Uncertainty

One way to obtain uncertainty estimates from a neural network is to use a probabilistic interpretation of the network output, $\hat{y} = lr(\mathbf{x};\theta)$. For instance, for classification problems, one can interpret the multi-dimensional output of a neural network as a logit and then apply the softmax function to obtain class probabilities. These probabilities can then be used to

construct common AL criteria, such as the maximum uncertainty, $\max_j \hat{y}^{(j)}$, or the predictive entropy, $-\sum_j \hat{y}^{(j)} \log \hat{y}^{(j)}$ [59]. Several works employ these classical criteria for neural AL [69, 52, 23].

Although interpreting the output of a neural network as a probabilistic quantity can be a useful way to obtain uncertainty information, it is not always reliable. For example, the network may exhibit high confidence in its prediction when it encounters out-of-distribution data points. In this case, the network's output probabilities may not accurately reflect its uncertainty, and using them for AL would not be ideal.

Epistemic Uncertainty

We may also use epistemic uncertainty measurements to perform AL, and this approach has been well-studied outside of neural AL. For example, regression and classification using Gaussian process are non-parametric Bayesian learning approaches with clear epistemic uncertainty measurements. We will cover AL methods for Gaussian process models in more detail in Subsection 2.3.3.

Capturing the epistemic uncertainty of a neural network for performing AL typically requires the use of a modified neural network that has a Bayesian interpretation as this provides a more concrete measure of epistemic uncertainty. Uncertainty quantification in neural networks has been extensively studied [1] and several methods, such as Bayesian neural networks and their approximations [39, 45, 16, 65], and deep ensemble techniques [35, 21], can be used to achieve this. These modified network models can be used to explicitly capture the uncertainty in predictions. One popular technique to capture the epistemic uncertainty of a neural network is Monte Carlo (MC) Dropout [16], which involves applying a Bernoulli distribution to the model weights using the Dropout layer, creating an approximate Bayesian neural networks which can be used with various acquisition functions from classical AL literature, such as maximum predictive entropy, variation ratios, and mean standard deviation of predictions. MC Dropout is a commonly used method in AL because it utilizes Dropout layers that are commonly used in deep learning, adding little additional modification during training and prediction.

Another class of AL algorithms uses an information theoretic approach for quantifying the uncertainty in model predictions. The Bayesian Active Learning by Disagreement (BALD) algorithm [25], for example, selects points that maximize the information gained about model parameters:

$$\alpha_{\text{BALD}}\left(\mathbf{x}, p\left(\theta|\mathbf{X}_L\right)\right) = \mathbb{I}\left[y, \theta|\mathbf{x}, \mathbf{X}_L\right] = \mathbb{H}\left[y|\mathbf{x}, \mathbf{X}_L\right] - \mathbb{E}_{p(\theta|\mathbf{X}_L)}\left[\mathbb{H}\left[y|\mathbf{x}, \theta\right]\right]$$

Inspecting the expression on the right, we see that the points that maximise the criterion are the ones at which the model prediction is uncertain (as indicated by the first term), but the model is expected to be certain over the random draws of parameters from the posterior distribution (as indicated by the second term). BALD was originally designed to acquire one data point in each step, following which the network was retrained. Since training takes

a long time, this obviously become a bottleneck for deep learning applications. To make it practically usable, the algorithm can be altered to select b points with the highest value of the criterion:

$$\alpha_{\text{BALD}}\left(\{\mathbf{x}_1, \ldots, \mathbf{x}_b\}, p\left(\theta|\mathbf{X}_L\right)\right) = \sum_{i=1}^{b} \mathbb{I}\left[y_i, \theta|\mathbf{x}_i, \mathbf{X}_L\right]$$

However, this does not consider the correlations between the individual points in a batch, and therefore leads to data inefficiency. BatchBALD [30] was introduced to address this problem, and it uses the mutual information between a batch of points (instead of a single point) and the model parameters:

$$\alpha_{\text{BatchBALD}}\left(\{\mathbf{x}_1, \ldots, \mathbf{x}_b\}, p\left(\theta|\mathbf{X}_L\right)\right) = \mathbb{I}\left[y_1, \ldots, y_b, \theta|\mathbf{x}_1, \ldots, \mathbf{x}_b, \mathbf{X}_L\right]$$

BatchBALD represents an enhancement over BALD as it considers the uncertainty within the batch, thereby preventing the selection of points that are too similar to one another.

Discussion

Uncertainty-based AL methods typically select points based on a measure of epistemic uncertainty, which allows them to provide better guarantees regarding the uncertainty of the trained model's predictions. However, a disadvantage of these methods is that they require network training since the information about the predictive uncertainty is obtained from studying the behavior of a model. In many works [25, 29, 30], for example, the uncertainty is captured by the MC Dropout technique [16] which requires network training in between batches. Consequently, these methods are much slower and very expensive to use. Although dropout is a commonly used technique in modern neural networks, it is desirable to perform AL without adding new layers to the model while still accurately capturing uncertainty.

In uncertainty-based methods, it is important to understand the source of randomness in the predictions. In methods based on MC Dropout, the parameters are assumed to be fixed, and the randomness arises from how the Dropout layers are set in a specific forward pass. However, these methods do not account for the fact that the final trained parameters are influenced by how the network is initialized, which introduces another source of randomness in the predictions. Therefore, to achieve initialization-robust training, relying solely on uncertainty from the Dropout layers may be insufficient.

2.2.3 Hybrid Strategies

Between the two extremes of considering solely the diversity of the input and solely predictive uncertainty, work has been done to take both these measures into account.

The Batch Active Learning by Diverse Gradient Embeddings (BADGE) algorithm [4] uses the hallucinated gradient space, or the gradient of the loss function with respect to the weights in the final layer and using the model's prediction, $g_\theta(\mathbf{x})$, as a proxy for the true

label, y:

$$h(\mathbf{x}) = \frac{\partial}{\partial \theta^L} \mathcal{L}\left(f(\mathbf{x}; \theta), g_\theta(\mathbf{x})\right)$$

The choice of using pseudo-labels makes the algorithm label-free and therefore, more applicable in real life settings. It can be shown that the gradient vectors are dependent on both the input data representation and also the data uncertainty as captured by the model's prediction. The algorithm then proceeds to select points that are highly disparate as well as high in magnitude, thereby ensuring that the active set selected in each round incorporates predictive uncertainty and sample diversity.

CLustering Uncertainty-weighted Embeddings (CLUE) [50] was designed for the problem of domain-adaptive AL, where the goal is to select data points which will result in good model generalization to some shifted target distribution. CLUE balances between selecting points with high data uncertainty and those which are diverse in the feature space by viewing the task as a weighted set-partitioning problem, where the uncertainty in the output of the penultimate layer of the network is used to weigh each data point.

2.2.4 Gaps in Literature

Thus far, we have examined the AL algorithms that have been introduced for neural networks. Diversity-based methods tend to prioritize selecting samples that are dissimilar from each other, rather than focusing on samples on which the model exhibits uncertainty. Additionally, they frequently necessitate the use of heuristics for measuring diversity but do not have a clear link to the predictive capacity of the trained models. In contrast, uncertainty-based algorithms can better capture model uncertainty in their selection criteria, but often require costly model training between batches or modifications to the model architecture to explicitly account for model uncertainty.

An additional constraint of many AL algorithms is that they require some labeled data before they can be utilized. For instance, in approaches where uncertainty or diversity is quantified using a trained model, a few labeled points are required for the initial round of training. This is frequently accomplished by randomly selecting the first batch. However, this might be unfavorable, especially when a party has a very limited budget and no data to begin with. In such situations, it is preferable to have an informed means of selecting the first batch instead of relying on random sampling.

An interesting gap within the AL literature, therefore, is to study uncertainty-based neural AL algorithms which require minimal rounds of training and which do not rely on an initial batch acquired through some other method. One way to address this concern is to develop AL methods that rely on a theoretical analysis of neural networks dynamics. Neural Tangent Kernels [26] can accurately model the output dynamics of a network, and may thus make a useful tool for the construction of an AL criterion that does not require any training yet can capture prediction uncertainty. We will discuss more about them in Section 2.4.

2.3 Gaussian Process

Definition 2.1. *For any set S, a Gaussian Process (GP) on S is a collection of random variables $(Z_t : t \in S)$ such that $\forall n \in \mathbb{N}$, $\forall t_1, \ldots, t_n \in S$, $(Z_{t_1}, \ldots, Z_{t_n})$ is multivariate Gaussian. In other words, any finite subset of this collection of random variables is Gaussian.*

Consider a dataset, $\mathcal{D} = (\mathbf{X}, \mathbf{y})$, where $\mathbf{X}$ is a collection of n vectorial inputs, $\mathbf{x}_i$, $\mathbf{y}$ is a collection of n corresponding noisy observations, $y_i = f(\mathbf{x}_i) + \xi_i$, f is the underlying (latent) function, and $\xi_i \sim \mathcal{N}(0, \sigma^2_{\text{noise}})$ is a Gaussian noise which we assume to be independent across different samples. Note that in this Bayesian setting, we assume that the function evaluations, $f(\mathbf{x})$, are Gaussian random variables $\forall \mathbf{x} \in \mathcal{X}$, and furthermore, $\{f(\mathbf{x})\}_{\mathbf{x}\in\mathcal{X}}$ is a GP over the input space.

For approximating the AL criterion in our work, we will utilise tools from the GP literature. We will briefly discuss the work related to GPs in this section, including full-rank computation, sparse approximations and AL for GP models.

2.3.1 Full-Rank GP

Gaussian process regression (GPR) is a non-parametric Bayesian framework that makes predictions incorporating prior knowledge (using kernels) and provides uncertainty measures over predictions. In GPR, one assumes a Gaussian prior over the data distribution given by mean $\mu(\cdot)$ and covariance matrix Σ. It is common to assume $\mu(\cdot) = 0$ for simplicity, and that the entries of Σ are given by some positive semi-definite (PSD) kernel, $\mathcal{K}(\cdot, \cdot)$. Consequently, we have

$$\mathbf{f} \sim \mathcal{N}(\mathbf{0}, \Sigma)$$

where $\mathbf{f} = [f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n)]^T$ and $\Sigma \in \mathbb{R}^{n\times n}$ is the covariance matrix such that $(\Sigma)_{ij} = \text{Cov}(f(\mathbf{x}_i), f(\mathbf{x}_j)) = \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)$. Given some data, $\mathcal{D} = (\mathbf{X}, \mathbf{y})$, using Bayes' Theorem, we can then show that

$$\mathbf{f}_T|\mathbf{y} \sim \mathcal{N}\left(\mathcal{K}_{\mathbf{X}_T\mathbf{X}}\left(\mathcal{K}_{\mathbf{X}} + \sigma^2_{\text{noise}} I\right)^{-1}\mathbf{y}, \mathcal{K}_{\mathbf{X}_T} - \mathcal{K}_{\mathbf{X}_T\mathbf{X}}\left(\mathcal{K}_{\mathbf{X}} + \sigma^2_{\text{noise}} I\right)^{-1}\mathcal{K}_{\mathbf{X}\mathbf{X}_T}\right) \tag{2.1}$$

where $\mathcal{K}_{\mathbf{X}} = \mathcal{K}(\mathbf{X}, \mathbf{X})$ and $\mathcal{K}_{\mathbf{X}\mathbf{X}_T} = \mathcal{K}^\top_{\mathbf{X}_T\mathbf{X}} = \mathcal{K}(\mathbf{X}, \mathbf{X}_T)$ [54].

Performing GPR, unfortunately, can be computationally expensive, especially for large datasets. This is because a matrix inversion is required for calculating the posterior covariance and this operation is $\mathcal{O}(n^3)$, where n is the number of points used to fit the model. Moreover, the exact posterior evaluation requires $\mathcal{O}(n^2)$ memory.

2.3.2 Sparse GP

Sparse approximations for GP were introduced as a way to approximate the true GP posterior. These techniques have reduced the time complexity for inference to $\mathcal{O}(nm^2)$ and the memory requirement to $\mathcal{O}(nm)$, for some fixed $m < n$. A baseline approximation for the full-rank GP could be to just work with a subset of m data points, instead of the whole

dataset of n points, to estimate the posterior. This would lower the computational complexity down to $\mathcal{O}(m^3)$, which is a significant improvement. However, it is obvious to see that this is not a very good strategy since we lose all the information in the points not included in the subset. Hence, in this review, we will focus on sparse methods that use m latent variables instead to approximate the posterior [51].

We consider m *inducing variables*, $\mathbf{u} = [y_1, \ldots, y_m]^T$ which are function evaluations at a set of m points, $\mathbf{U} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\} \subset \mathcal{X}$, which we call the *inducing points*. We can then express the joint distribution of $\mathbf{f}$ and $\mathbf{f}_T$ as

$$p(\mathbf{f}, \mathbf{f}_T) = \int p(\mathbf{f}, \mathbf{f}_T, \mathbf{u})\, d\mathbf{u} = \int p(\mathbf{f}, \mathbf{f}_T|\mathbf{u})\, p(\mathbf{u})\, d\mathbf{u}$$

where $\mathbf{u} \sim \mathcal{N}(\mathbf{0}, \mathcal{K}_{\mathbf{U}})$. This is an exact expression for the joint distribution and we can approximate it by assuming conditional independence between $\mathbf{f}$ and $\mathbf{f}_T$ given $\mathbf{u}$:

$$p(\mathbf{f}, \mathbf{f}_T) \approx q(\mathbf{f}, \mathbf{f}_T) = \int q(\mathbf{f}|\mathbf{u})\, q(\mathbf{f}_T|\mathbf{u})\, p(\mathbf{u})\, d\mathbf{u} \tag{2.2}$$

The exact conditional distributions can be calculated as noise-less versions of GP posterior given in Equation 2.1:

$$\begin{aligned} \mathbf{f}|\mathbf{u} &\sim \mathcal{N}\left(\mathcal{K}_{\mathbf{XU}}\mathcal{K}_{\mathbf{U}}^{-1}\mathbf{u}, \mathcal{K}_{\mathbf{X}} - \mathcal{Q}_{\mathbf{X}}\right) \\ \mathbf{f}_T|\mathbf{u} &\sim \mathcal{N}\left(\mathcal{K}_{\mathbf{X}_T\mathbf{U}}\mathcal{K}_{\mathbf{U}}^{-1}\mathbf{u}, \mathcal{K}_{\mathbf{X}_T} - \mathcal{Q}_{\mathbf{X}_T}\right) \end{aligned} \tag{2.3}$$

where we define $\mathcal{Q}_{\mathbf{AB}} = \mathcal{K}_{\mathbf{AU}}\mathcal{K}_{\mathbf{U}}^{-1}\mathcal{K}_{\mathbf{UB}}$ and $\mathcal{Q}_{\mathbf{A}} = \mathcal{Q}_{\mathbf{AA}}$. Now, we discuss sparse methods that approximate the conditionals in Equation 2.3.

Subset of Regressors

The Subset of Regressors (SoR) algorithm [63] is a linear model with a particular prior on the weights:

$$\mathbf{w}_{\mathbf{U}} \sim \mathcal{N}\left(\mathbf{0}, \mathcal{K}_{\mathbf{U}}^{-1}\right)$$

so that for any $\mathbf{x} \in \mathcal{X}$ we have

$$f(\mathbf{x}) = \mathcal{K}_{\mathbf{xU}}\mathbf{w}_{\mathbf{U}} \tag{2.4}$$

This set up allows us to recover the exact GP prior on the inducing variables, $\mathbf{u} = \mathcal{K}_{\mathbf{U}}\mathbf{w}_{\mathbf{U}}$:

$$\begin{aligned} \mathbb{E}[\mathbf{u}] &= \mathcal{K}_{\mathbf{U}}\mathbb{E}[\mathbf{w}_{\mathbf{U}}] = \mathbf{0} \\ \mathrm{Cov}(\mathbf{u}) &= \mathbb{E}\left[\mathbf{u}\mathbf{u}^T\right] = \mathcal{K}_{\mathbf{U}}\mathbb{E}\left[\mathbf{w}_{\mathbf{U}}\mathbf{w}_{\mathbf{U}}^T\right]\mathcal{K}_{\mathbf{U}} = \mathcal{K}_{\mathbf{U}} \end{aligned}$$

so that $\mathbf{u} \sim \mathcal{N}(\mathbf{0}, \mathcal{K}_{\mathbf{U}})$. Now, we can substitute $\mathbf{w}_{\mathbf{U}} = \mathcal{K}_{\mathbf{U}}^{-1}\mathbf{u}$ in Equation 2.4 to get

$$f(\mathbf{x}) = \mathcal{K}_{\mathbf{xU}}\mathcal{K}_{\mathbf{U}}^{-1}\mathbf{u}$$

This implies that there is a deterministic relationship between $f(\mathbf{x})$ and $\mathbf{u}$, and so, the approximate conditionals in Equation 2.2 can be written as

$$\mathbf{f}|\mathbf{u} \overset{\text{SoR}}{\sim} \mathcal{N}(\mathcal{K}_{\mathbf{XU}}\mathcal{K}_{\mathbf{U}}^{-1}\mathbf{u}, \mathbf{0})$$

Note that the SoR model implies a zero conditional covariance on the prior. We can now write the approximate joint prior from Equation 2.2 as

$$q(\mathbf{f}, \mathbf{f}_T) = \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathcal{Q}_{\mathbf{X}} & \mathcal{Q}_{\mathbf{X}\mathbf{X}_T} \\ \mathcal{Q}_{\mathbf{X}_T\mathbf{X}} & \mathcal{Q}_{\mathbf{X}_T} \end{bmatrix}\right) \tag{2.5}$$

Under the SoR model, the posterior distribution is given as

$$\mathbf{f}_T|\mathbf{y} \overset{\text{SoR}}{\sim} \mathcal{N}(\sigma_{\text{noise}}^{-2}\mathcal{K}_{\mathbf{X}_T\mathbf{U}}\mathcal{S}_{\mathbf{XU}}\mathcal{K}_{\mathbf{UX}}\mathbf{y}, \mathcal{K}_{\mathbf{X}_T\mathbf{U}}\mathcal{S}_{\mathbf{XU}}\mathcal{K}_{\mathbf{U}\mathbf{X}_T}) \tag{2.6}$$

where $\mathcal{S}_{\mathbf{XU}} = \left(\sigma_{\text{noise}}^{-2}\mathcal{K}_{\mathbf{UX}}\mathcal{K}_{\mathbf{XU}} + \mathcal{K}_{\mathbf{U}}\right)^{-1}$ is equal to the posterior on the model weights, $\mathbf{w}_{\mathbf{U}}$. This approximation is equivalent to performing exact GP inference with the degenerate kernel function

$$\mathcal{K}_{\text{SoR}}(\mathbf{x}, \mathbf{x}') = \mathcal{K}(\mathbf{x}, \mathbf{u})\mathcal{K}_{\mathbf{U}}^{-1}\mathcal{K}(\mathbf{u}, \mathbf{x}')$$

The SoR model is much cheaper than performing full-rank GP inference – the computational complexity for fitting to the labeled dataset is now $\mathcal{O}(nm^2)$, and $\mathcal{O}(m)$ and $\mathcal{O}(m^2)$ for calculating the posterior mean and variance of each test point, respectively.

Unfortunately, this approximation is quite restrictive since the GP prior implied by it has only m degrees of freedom and is thus, degenerate. Since only m linearly independent functions can be sampled from the prior, the posterior is severely constrained even with a small number of data points. This leads the model to often make nonsensically overconfident predictions, which is a common problem for finite linear models [53].

Deterministic Training Conditional

To tackle the nonsensical predictive uncertainties of the SoR model, [57] introduced the Projected Latent Variables (PLV) model, which assumes the projection $\mathbf{f} = \mathcal{K}_{\mathbf{XU}}\mathcal{K}_{\mathbf{U}}^{-1}\mathbf{u}$, so that

$$\mathbf{y}|\mathbf{f} \underset{\text{DTC}}{\overset{d}{=}} \mathbf{y}|\mathbf{u} \sim \mathcal{N}\left(\mathcal{K}_{\mathbf{XU}}\mathcal{K}_{\mathbf{U}}^{-1}\mathbf{u}, \sigma_{\text{noise}}^2 I\right)$$

An alternate formulation of this assumption would be to approximate the training conditional as

$$\mathbf{f}|\mathbf{u} \overset{\text{DTC}}{\sim} \mathcal{N}\left(\mathcal{K}_{\mathbf{XU}}\mathcal{K}_{\mathbf{U}}^{-1}\mathbf{u}, \mathbf{0}\right)$$

which gives it the name Deterministic Training Conditional (DTC) approximation. Note that the training conditional for the DTC model is the same as for the SoR model (Equation 2.3). The DTC model, however, differs in its use of the exact test conditional instead of an approximation:

$$q(\mathbf{f}_T|\mathbf{u}) = p(\mathbf{f}_T|\mathbf{u})$$

This reformulation allows us to perform exact inference, however, using approximate priors. We can now write the approximate joint prior from Equation 2.2 as

$$q\left(\mathbf{f}, \mathbf{f}_T\right) = \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathcal{Q}_{\mathbf{X}} & \mathcal{Q}_{\mathbf{X}\mathbf{X}_T} \\ \mathcal{Q}_{\mathbf{X}_T\mathbf{X}} & \mathcal{K}_{\mathbf{X}_T} \end{bmatrix}\right)$$

This approximation is very similar to the one given by the SoR model in Equation 2.5, and the difference comes from the exact formulation of the test conditional. Under the DTC approximation, the posterior distribution is given as

$$\mathbf{f}_T|\mathbf{y} \overset{\text{DTC}}{\sim} \mathcal{N}(\sigma_{\text{noise}}^{-2}\mathcal{K}_{\mathbf{X}_T\mathbf{U}}\mathcal{S}_{\mathbf{X}\mathbf{U}}\mathcal{K}_{\mathbf{U}\mathbf{X}}\mathbf{y}, \mathcal{K}_{\mathbf{X}_T} - \mathcal{Q}_{\mathbf{X}_T} + \mathcal{K}_{\mathbf{X}_T\mathbf{U}}\mathcal{S}_{\mathbf{X}\mathbf{U}}\mathcal{K}_{\mathbf{U}\mathbf{X}_T})$$

where $\mathcal{S}_{\mathbf{X}\mathbf{U}}$ is as defined in Equation 2.6. The posterior mean given by the DTC model is the same as given by the SoR model (Equation 2.6), but the covariance has an additional term, $\mathcal{K}_{\mathbf{X}_T} - \mathcal{Q}_{\mathbf{X}_T}$, that accounts for the reformulation. Note that $\mathcal{K}_{\mathbf{X}_T} - \mathcal{Q}_{\mathbf{X}_T}$ is positive definite, and hence, the variances at individual test points as predicted by the DTC model are higher than as predicted by the SoR model. This addresses the problem of making erroneously overconfident predictions.

It is interesting to note that, unlike in case of the SoR approximation, the DTC model does not correspond exactly to a GP. This is because the covariance values depend on whether the points have been included in the training set or not, thus violating consistency. The computational complexity is the same as for the SoR model.

Fully-Independent Training Conditional

The Sparse Gaussian Processes using Pseudo-inputs (SGPP) approximation [64] uses a richer covariance function than the DTC model, and proposes a more sophisticated likelihood approximation:

$$\mathbf{y}|\mathbf{f} \underset{\text{FITC}}{\overset{d}{=}} \mathbf{y}|\mathbf{u} \sim \mathcal{N}\left(\mathcal{K}_{\mathbf{X}\mathbf{U}}\mathcal{K}_{\mathbf{U}}^{-1}\mathbf{u}, \operatorname{diag}\left[\mathcal{K}_{\mathbf{X}} - \mathcal{Q}_{\mathbf{X}}\right] + \sigma_{\text{noise}}^2 I\right)$$

where $\operatorname{diag}[A]$ is defined as the diagonal matrix with entries matching that of the diagonal of A. An alternate formulation of this assumption would be to approximate the training conditional as

$$\mathbf{f}|\mathbf{u} \overset{\text{FITC}}{\sim} \mathcal{N}\left(\mathcal{K}_{\mathbf{X}\mathbf{U}}\mathcal{K}_{\mathbf{U}}^{-1}\mathbf{u}, \operatorname{diag}\left[\mathcal{K}_{\mathbf{X}} - \mathcal{Q}_{\mathbf{X}}\right]\right) \quad \text{and} \quad q\left(\mathbf{f}_T|\mathbf{u}\right) = p\left(\mathbf{f}_T|\mathbf{u}\right)$$

Unlike SoR and DTC, FITC does not assume a deterministic relationship between $\mathbf{f}$ and $\mathbf{u}$, but instead imposes a conditional independence assumption:

$$q\left(\mathbf{f}|\mathbf{u}\right) = \prod_{i=1}^{n} p\left(f_i|\mathbf{u}\right)$$

which gives it the name Fully-Independent Training Conditional (FITC) approximation. Moreover, as in DTC, the exact test conditional distribution (Equation 2.3) is used. For a

single test case, the approximate joint prior implied by FITC is given as

$$q(\mathbf{f}, f_T) = \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathcal{Q}_{\mathbf{X}} + \operatorname{diag}\left[\mathcal{K}_{\mathbf{X}} - \mathcal{Q}_{\mathbf{X}}\right] & \mathcal{Q}_{\mathbf{X}\mathbf{x}} \\ \mathcal{Q}_{\mathbf{x}\mathbf{X}} & \mathcal{K}_{\mathbf{x}} \end{bmatrix}\right)$$

FITC improves on DTC by replacing the variances of the training points with the exact values. Under the FITC approximation, the posterior distribution is given as

$$q_{\text{FITC}}(f_T|\mathbf{y}) = \mathcal{N}\left(\mathcal{K}_{\mathbf{x}\mathbf{U}}\mathcal{S}_{\mathbf{X}\mathbf{U}}\mathcal{K}_{\mathbf{U}\mathbf{X}}\Lambda^{-1}\mathbf{y}, \mathcal{K}_{\mathbf{x}} - \mathcal{Q}_{\mathbf{x}} + \mathcal{K}_{\mathbf{x}\mathbf{U}}\mathcal{S}_{\mathbf{X}\mathbf{U}}\mathcal{K}_{\mathbf{U}\mathbf{x}}\right)$$

where we have redefined $\mathcal{S}_{\mathbf{X}\mathbf{U}} = \left(\mathcal{K}_{\mathbf{U}} + \mathcal{K}_{\mathbf{U}\mathbf{X}}\Lambda^{-1}\mathcal{K}_{\mathbf{X}\mathbf{U}}\right)^{-1}$ and introduced $\Lambda = \operatorname{diag}\left[\mathcal{K}_{\mathbf{X}} - \mathcal{Q}_{\mathbf{X}} + \sigma^2_{\text{noise}}I\right]$. The computational complexity is the same as for SoR and DTC. For more than one test case, we can do one of two things:

1. use the exact full test conditional as in Equation 2.3, but then the FITC approximation would not correspond to an exact GP since the covariance calculation is different for training and test points

2. extend the conditional independence assumption to the test points, in which case the Fully Independent Conditional (FIC) approximation is equivalent to performing exact GP inference with the non-degenerate kernel function

   $$\mathcal{K}_{\text{FIC}}(\mathbf{x}, \mathbf{x}') = \mathcal{K}_{\text{SoR}}(\mathbf{x}, \mathbf{x}') + \delta(\mathbf{x}, \mathbf{x}')\left[\mathcal{K}(\mathbf{x}, \mathbf{x}') - \mathcal{K}_{\text{SoR}}(\mathbf{x}, \mathbf{x}')\right]$$

   where $\delta(\cdot, \cdot)$ is the Kronecker delta function. The effective prior implied by FIC is

   $$q(\mathbf{f}, f_T) = \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathcal{Q}_{\mathbf{X}} + \operatorname{diag}\left[\mathcal{K}_{\mathbf{X}} - \mathcal{Q}_{\mathbf{X}}\right] & \mathcal{Q}_{\mathbf{X}\mathbf{x}} \\ \mathcal{Q}_{\mathbf{x}\mathbf{X}} & \mathcal{Q}_{\mathbf{x}} + \operatorname{diag}\left[\mathcal{K}_{\mathbf{x}} - \mathcal{Q}_{\mathbf{x}}\right] \end{bmatrix}\right)$$

2.3.3 AL for GP

The problem of AL has also been extensively studied within the GP literature. This problem is similar to Bayesian optimization, which is a technique of zero-order optimization, but differs in that in AL we are more interested in learning the overall function rather than just the optimal point for the function. From the point of view of Bayesian optimization, AL can be thought of as focusing on the exploration aspect and not on the exploitation.

Due to the Bayesian nature of GPs, it is simple to use the uncertainty measures in selection of an active set. Similar to the AL problem as described above, the goal is to select points which will give as much information about the global distribution as possible and therefore, the selected points should cover the input space well, while also not be too similar to each other. [32] suggested that the mutual information criterion between the active set and the remaining data is a useful criterion. It has also been shown that greedily selecting points which maximise the mutual information is a good enough strategy due to the submodularity of mutual information.

2.4 Neural Tangent Kernel

In order to choose data points that, when used to train the network, will ensure initialization robustness, we need to examine how the network will evolve under gradient descent. A notable work within this area is based on the theory of neural tangent kernels.

2.4.1 Network Parameterization

First off, we work with a particular parameterization of neural networks as defined in [26].

Definition 2.2. *Define $f(\cdot;\theta)$ as a multilayer perceptron with L hidden layers of widths $k_1, k_2, \ldots, k_L$. Let the input dimension of the network be k_0 and the output dimension k_{L+1}. Let the network be parameterized as*

$$\begin{aligned} h^{(0)}(x) &= x \\ h^{(l)}(x) &= \phi\left(\frac{1}{\sqrt{k_l}} W^{(l)} h^{(l-1)}(x) + b^{(l)}\right), \quad \forall l \in [1, \ldots, L] \\ f(x) &= W^{(L+1)} h^{(L)}(x) + b^{(L+1)} \end{aligned}$$

where $W_{ij}^{(l)} \sim \mathcal{N}(0, \sigma_W^2)$ and $b_{ij}^{(l)} \sim \mathcal{N}(0, \sigma_b^2)$ are model weights and biases initialized randomly from a Gaussian distribution with variances σ_W^2 and σ_b^2 respectively, and $\phi : \mathbb{R} \to \mathbb{R}$ is a Lipschitz twice differentiable non-linearity function with bounded second derivative.

Note that the model is parameterized differently from usual due to the $1/\sqrt{k_l}$ factors. This is for making proofs in the asymptotic case ($n_1, \ldots, n_{L-1} \to \infty$) convenient. However, the initialization of the neural network is still based on sampling from a Gaussian distribution which is comparable to classical initialization methods.

We will write the model parameters as $\theta = \text{flatten}\left(W^{(1)}, b^{(1)}, \ldots, W^{(L)}, b^{(L)}\right)$ and $\theta^{\leq l} = \text{flatten}\left(W^{(1)}, b^{(1)}, \ldots, W^{(l)}, b^{(l)}\right)$. We will also sometimes use $f(x;\theta)$ to denote the model output, $f(x)$, explicitly stating the dependence on the parameters, θ. The model output is given by $\hat{y} = f(x;\theta)$, and the model is trained with a loss function, $\mathcal{L}(\hat{y}, y)$, where y is the true label corresponding to the input x.

2.4.2 Training Dynamics

Suppose we have a neural network whose parameters are given by θ. We would like to train it using gradient descent, i.e., using the update formula $\theta_{t+1} \leftarrow \theta_t - \eta \cdot \nabla_\theta \mathcal{L}$, where η is the learning rate. For a small enough learning rate, we can approximate this process as gradient flow, given by the differential equation

$$\dot{\theta}_t = -\eta \nabla_\theta \mathcal{L} = -\eta \nabla_\theta f_t(\mathbf{X})^\top \nabla_{f_t} \mathcal{L} \tag{2.7}$$

We can then write the change in the function prediction as

$$\dot{f}_t(\mathbf{X}) = \nabla_\theta f_t(\mathbf{X})\, \dot{\theta}_t = -\eta \nabla_\theta f_t(\mathbf{X})\, \nabla_\theta f_t(\mathbf{X})^\top \nabla_{f_t} \mathcal{L} \tag{2.8}$$

The term $\hat{\Theta}_t(\mathbf{X}, \mathbf{X}') \triangleq \nabla_\theta f_t(\mathbf{X}) \nabla_\theta f_t^\top(\mathbf{X}')$ is referred to as the (empirical) neural tangent kernel (NTK) [26, 36]. In words, this kernel is the outer product of the gradients of the model outputs with respect to its weights. At finite-width, it will depend on the specific random draw of the parameters.

The NTK has interesting properties for wide neural networks, i.e., when

$$\min_{l \in \{1,2,\ldots,L\}} k_l \to \infty$$

When a neural network is wide enough, it can be approximated as a linear model [36],

$$f(\mathbf{x}; \theta) \approx f^{\text{lin}}(\mathbf{x}; \theta) = f(\mathbf{x}; \theta_0) + \langle \nabla_\theta f(\mathbf{x}; \theta_0), \theta - \theta_0 \rangle$$

where $\theta_0 \sim \text{init}(\theta)$. The condition for linearity can be further relaxed to when the Hessian norm of the network approaches 0 as the network width increases [37].

In the infinite width setting, empirical NTK at initialization converges in probability to a deterministic kernel, $\hat{\Theta}_0 \xrightarrow{\mathcal{P}} \Theta$, which is referred to as the analytical kernel [36]. This kernel stays constant during the training process. We will overload notation sometimes and write $\Theta_{XX'} \coloneqq \Theta(X, X')$ and $\Theta_X \coloneqq \Theta_{XX}$.

A linearized neural network is a useful approximation since, assuming the model is trained using squared-error loss

$$\mathcal{L}(\mathbf{x}, y; \theta) = \frac{1}{2} \| f(\mathbf{x}; \theta) - y \|^2, \tag{2.9}$$

the gradient of the loss is given by $\nabla_\theta \mathcal{L}_{f^{\text{lin}}} = \nabla_\theta f(\mathbf{x}; \theta_0)$ and a closed-form solution of Equation 2.7 and Equation 2.8 can be computed. Under this approximation, for a randomly initialized neural network trained on the data $\mathcal{D} = (\mathbf{X}, \mathbf{y})$ until convergence, the prediction of the converged model will follow the distribution

$$f(\mathbf{X}_T) | \mathbf{y} \sim \mathcal{N}\left(\mu_{\text{NN}}(\mathbf{X}_T | \mathcal{D}), \Sigma_{\text{NN}}(\mathbf{X}_T | \mathcal{D})\right), \tag{2.10}$$

where the predictive mean is given by

$$\mu_{\text{NN}}(\mathbf{X}_T | \mathbf{y}) = \Theta_{\mathbf{X}_T \mathbf{X}} \Theta_{\mathbf{X}}^{-1} \mathbf{y},$$

and the predictive covariance by

$$\begin{aligned} \Sigma_{\text{NN}}(\mathbf{X}_T | \mathbf{y}) = \mathcal{K}_{\mathbf{X}_T} &+ \Theta_{\mathbf{X}_T \mathbf{X}} \Theta_{\mathbf{X}}^{-1} \mathcal{K}_{\mathbf{X}} \Theta_{\mathbf{X}}^{-1} \Theta_{\mathbf{X} \mathbf{X}_T} \\ &- \left(\Theta_{\mathbf{X}_T \mathbf{X}} \Theta_{\mathbf{X}}^{-1} \mathcal{K}_{\mathbf{X} \mathbf{X}_T} + \mathcal{K}_{\mathbf{X}_T \mathbf{X}} \Theta_{\mathbf{X}}^{-1} \Theta_{\mathbf{X} \mathbf{X}_T} \right), \end{aligned} \tag{2.11}$$

where the kernel

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = \mathbb{E}_{\theta_0 \sim \text{init}(\theta)} \left[f(\mathbf{x}; \theta_0) \cdot f(\mathbf{x}'; \theta_0) \right]$$

is the covariance of the model output with respect to the random initialization [36]. In the case where model regularization is added to the loss function, we can replace $\Theta_{\mathbf{X}}$ and $\mathcal{K}_{\mathbf{X}}$ with

$\Theta_{\mathbf{X}} + \lambda^2 I$ and $\mathcal{K}_{\mathbf{X}} + \lambda^2 I$, respectively. From this, we are able to see the converged neural network has predictive mean equivalent to that of kernel regression, and has predictive covariance which depends on the model architecture and the input data. This result is important since it allows us to understand how the randomness in initialization affects the final predictive distribution. Unfortunately, a similar closed-form distribution cannot be formulated for neural networks which are trained under a different objective, such as the cross-entropy loss, which is commonly used in classification problems. The best that one can hope to do is to solve Equation 2.7 and Equation 2.8 numerically.

2.4.3 Approximating Σ_{NN} using GP Posterior

Unfortunately, one issue with this criterion is that the covariance matrix Σ_{NN} is expensive to compute. We therefore, propose to consider, as a proxy, a covariance which shares similarity with that in GP instead.

If we were to perform GPR (Subsection 2.3.1) on some data assuming the NTK as the covariance function, which we will call the NTKGP [21], then the predictive distribution will be $f(\mathbf{X}_T|\mathbf{y}) \sim \mathcal{GP}(\mu_{\text{NTKGP}}(\mathbf{X}_T|\mathbf{y}), \Sigma_{\text{NTKGP}}(\mathbf{X}_T|\mathbf{y}))$, where

$$\begin{aligned} \mu_{\text{NTKGP}}(\mathbf{X}_T|\mathbf{y}) &= \Theta_{\mathbf{X}_T\mathbf{X}}\Theta_{\mathbf{X}}^{-1}\mathbf{y} \\ \Sigma_{\text{NTKGP}}(\mathbf{X}_T|\mathbf{y}) &= \Theta_{\mathbf{X}_T} - \Theta_{\mathbf{X}_T\mathbf{X}}\Theta_{X}^{-1}\Theta_{\mathbf{X}\mathbf{X}_T}. \end{aligned} \tag{2.12}$$

The covariance function, Σ_{NTKGP}, is simpler to work with for two reasons:

- Only the NTK, Θ, needs to be computed, and not the kernel $\mathcal{K}$. Furthermore, NTK is simple to approximate since it is defined as the inner product of the model output gradients with respect to the parameters.
- The posterior distribution of GPs are well-studied, and there are many tools available for approximating it, like those introduced in Subsection 2.3.2.

2.4.4 Applications

Due to the guarantees on predictive distribution, NTK is a useful method of obtaining uncertainty measurements from neural networks. For example, [21] provides a method of augmenting a neural network such that training it using gradient descent is equivalent to drawing a sample from some GP posterior distribution. This is useful since it is no longer necessary to invert a large matrix in order to compute the posterior distribution and therefore, it allows a more convenient method for performing Bayesian inference.

The measurements in uncertainty that the NTK provides are also useful for data valuation. For example, [71] uses NTKs to value a dataset based on the expected generalization loss after training. A benefit of such method is that data valuation can be done at the model initialization stage, without having to perform any actual model training.

In a similar vein to data valuation, NTKs can also be used in picking useful data points for AL problems. [70] presents a streaming-based AL algorithm which selects points based on the uncertainty in the trained network's predictions. It selects query points based on the generalization loss from the NTK theory, however only provides limited empirical results and limited theoretical guarantee when the points are selected in batch mode. The most notable attempt in applying NTKs to AL is the Most Likely Model Output Change (MLMOC) algorithm [42] which uses linearized models to predict how a network will behave if some data point was added to the training set.

2.4.5 Limitations

As mentioned, the theory of NTK assumes a network with NTK parameterization and whose hidden layer widths approach infinity. [2] has established an error bound on the approximation of the NTK in terms of the network width. In practice, many AL algorithms based on NTKs calculate their criterion using the empirical NTK of the finite-width neural networks in use, yet they record good performance. For most choices of (σ_b, σ_W), the NTK converges exponentially to a constant as the network depth increases [20]. This is another limitation of the NTK regime – it will break down as the network gets too deep.

Chapter 3

Theory

In this chapter, we establish the theoretical results of our work. First off, in Section 3.1 we use the theory of NTKs, as described in Section 2.4, to model the variance in the neural network output w.r.t. initialization. We then use tools from linear algebra and sparse GP literature, as discussed in Subsection 2.3.2, to introduce a computationally efficient approximation of this quantity. Furthermore, in Section 3.2, we derive a theoretical guarantee on quality of this approximation. In Section 3.3, we show that the approximation can also be used to bound the generalization error of the network at individual test points. Finally, in Section 3.4, we construct an AL criterion using the approximated output variance.

3.1 Output Variance of the Trained Network

In Subsection 2.4.2, we saw that if an infinite width neural network with initial parameters θ_0 is trained till convergence, then the predictions of the converged network at the test points follow a GP (Equation 2.10), where the randomness comes from the initialization of θ_0 [36]. Hence, we can use Σ_{NN} (Equation 2.11) as a natural and principled measure for initialization robustness. Unfortunately, Σ_{NN} requires the computation of two different kernels, Θ and $\mathcal{K}$, and it involves multiple matrix operations and the inversion of the $n \times n$ NTK of the training set. This puts the computation complexity of Σ_{NN} at $\mathcal{O}\left(n^3\right)$, making it unusable in practical settings.

To approximate the output variance efficiently, we can replace the covariance matrix, Σ_{NN}, with Σ_{NTKGP} (Equation 2.12). This would reduce the number of matrix computations to just one – the NTK, Θ. Even though the computation of Σ_{NTKGP} still incurs a cost of $\mathcal{O}\left(n^3\right)$ due to the inversion of Θ, we can reduce this cost down to $\mathcal{O}\left(n^2\right)$ by performing low-rank updates in each round of AL. We can further reduce the dependence on n using sparse GP techniques discussed in Subsection 2.3.2.

3.2 Approximation Quality

Since $\Sigma_{\text{NN}}(\mathbf{X}_T|\mathcal{D}) \preceq \Sigma_{\text{NTKGP}}(\mathbf{X}_T|\mathcal{D})$ [21], we have that for a single test point,

$$\sigma^2_{\text{NN}}(\mathbf{x}_T|\mathcal{D}) \leq \sigma^2_{\text{NTKGP}}(\mathbf{x}_T|\mathcal{D})$$

where $\sigma^2_{\text{NN}}(\mathbf{x}_T|\mathcal{D}) \coloneqq \Sigma_{\text{NN}}(\mathbf{x}_T|\mathcal{D})$ and $\sigma^2_{\text{NTKGP}}(\mathbf{x}_T|\mathcal{D}) \coloneqq \Sigma_{\text{NTKGP}}(\mathbf{x}_T|\mathcal{D})$. Therefore, we have that $\sigma^2_{\text{NTKGP}}(\mathbf{x}_T|\mathcal{D})$ overestimates the true output variance, $\sigma^2_{\text{NN}}(\mathbf{x}_T|\mathcal{D})$. To theoretically justify our choice of approximating Σ_{NN} with Σ_{NTKGP}, we will derive a bound on the approximation error for neural networks with ReLU activation.

3.2.1 Dual Activations

We first recall the concept of dual activations, as introduced in [13, 36].

Definition 3.1. *The dual of an activation function $\phi : \mathbb{R} \to \mathbb{R}$ is a real-valued function defined on the space of 2×2 positive semi-definite matrices,*

$$\tilde{\phi}(\Lambda) = \mathbb{E}_{(u,v)\sim\mathcal{N}(0,\Lambda)}[\phi(u)\,\phi(v)]$$

Similarly, if we let ϕ' be the derivative of ϕ, then its dual is defined as

$$\tilde{\phi'}(\Lambda) = \mathbb{E}_{(u,v)\sim\mathcal{N}(0,\Lambda)}[\phi'(u)\,\phi'(v)]$$

3.2.2 Network Parameterization

Consistent with [36], we work with the network parameterization as described in Subsection 2.4.1. Furthermore, we assume that the activation function, ϕ, is scaled so that

$$\tilde{\phi}\left(\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}\right) = 1 \tag{3.1}$$

In the later sections, we will see that this assumption will help simplify the repeated application of the activation function to a given input.

3.2.3 Relationship between $\mathcal{K}$ and Θ

Since Σ_{NTKGP} is obtained by replacing $\mathcal{K}$ with Θ in the expression for Σ_{NN}, it is interesting to note the relationship between the two kernels. The following lemma is taken from [26].

Lemma 3.1. *For a neural network in the infinite-width regime, $\mathcal{K}$ and Θ can be defined*

recursively as

$$\mathcal{K}^{(0)}(\mathbf{x},\mathbf{x}') = \mathbf{x}^T\mathbf{x}' + \sigma_b^2$$
$$\Theta^{(0)}(\mathbf{x},\mathbf{x}') = 0$$
$$\mathcal{K}^{(l)}(\mathbf{x},\mathbf{x}') = \tilde{\phi}\left(\begin{bmatrix}\mathcal{K}^{(l-1)}(\mathbf{x},\mathbf{x}) & \mathcal{K}^{(l-1)}(\mathbf{x},\mathbf{x}') \\ \mathcal{K}^{(l-1)}(\mathbf{x}',\mathbf{x}) & \mathcal{K}^{(l-1)}(\mathbf{x}',\mathbf{x}')\end{bmatrix}\right) + \sigma_b^2, \qquad l \in \{1,\cdots,L+1\} \tag{3.2}$$
$$\Theta^{(l)}(\mathbf{x},\mathbf{x}') = \mathcal{K}^{(l)}(\mathbf{x},\mathbf{x}') + \Theta^{(l-1)}(\mathbf{x},\mathbf{x}')\cdot\dot{\mathcal{K}}^{(l-1)}(\mathbf{x},\mathbf{x}'), \qquad l \in \{1,\cdots,L+1\} \tag{3.3}$$

where we define

$$\dot{\mathcal{K}}^{(l)}(\mathbf{x},\mathbf{x}') = \tilde{\phi}'\left(\begin{bmatrix}\mathcal{K}^{(\ell)}(\mathbf{x},\mathbf{x}) & \mathcal{K}^{(\ell)}(\mathbf{x},\mathbf{x}') \\ \mathcal{K}^{(\ell)}(\mathbf{x}',\mathbf{x}) & \mathcal{K}^{(\ell)}(\mathbf{x}',\mathbf{x}')\end{bmatrix}\right) \tag{3.4}$$

Notice that Equation 3.3 gives a recursive formula for computing Θ. If we "unroll" this formula, we will obtain

$$\Theta(\mathbf{x},\mathbf{x}') = \sum_{l=1}^{L+1}\mathcal{K}^{(l)}(\mathbf{x},\mathbf{x}')\prod_{l'=l+1}^{L+1}\dot{\mathcal{K}}^{(l')}(\mathbf{x},\mathbf{x}') \tag{3.5}$$

Given Equation 3.5, to bound $\Theta(\mathbf{x},\mathbf{x}')$, we simply need to provide bounds for each of $\mathcal{K}^{(l)}$ and $\dot{\mathcal{K}}^{(l)}$ individually. This can be done by inspecting the dual activation functions.

3.2.4 ReLU Dual Activation Function

We will rescale the classical ReLU activation function as $\phi(x) = \sqrt{2}\max(x,0)$ so that its dual satisfies Equation 3.1. [36] showed that for ReLU activations, if $\Lambda = \begin{bmatrix}\mathbf{x}^T\mathbf{x} & \mathbf{x}^T\mathbf{x}' \\ \mathbf{x}'^T\mathbf{x} & \mathbf{x}'^T\mathbf{x}'\end{bmatrix}$, then

$$\tilde{\phi}(\Lambda) = \frac{1}{\pi}\|\mathbf{x}\|_2\|\mathbf{x}'\|_2(\sin\theta + (\pi-\theta)\cos\theta), \quad \text{and}$$
$$\tilde{\phi}'(\Lambda) = 1 - \frac{\theta}{\pi}, \quad \text{where}$$
$$\theta = \cos^{-1}\left(\frac{\mathbf{x}^T\mathbf{x}'}{\|\mathbf{x}\|\|\mathbf{x}'\|}\right)$$

For convenience, define the function $\rho : [-1,1] \to \mathbb{R}$, where

$$\rho(r) = \tilde{\phi}\left(\begin{bmatrix}1 & r \\ r & 1\end{bmatrix}\right) = \frac{1}{\pi}\left(\sqrt{1-r^2} + r\left(\pi - \cos^{-1}r\right)\right) \tag{3.6}$$

This can be thought of as a re-parametrization of the dual activation function $\tilde{\phi}$ in the case where $\|\mathbf{x}\| = \|\mathbf{x}'\| = 1$, and $r = \cos\theta$ is the cosine distance between $\mathbf{x}$ and $\mathbf{x}'$. We can also define ρ' similarly but on the dual activation $\tilde{\phi}'$ instead,

$$\rho'(r) = \tilde{\phi}'\left(\begin{bmatrix}1 & r \\ r & 1\end{bmatrix}\right) = 1 - \frac{\cos^{-1}r}{\pi} \tag{3.7}$$

Using these definitions, we can verify that

$$\tilde{\phi}(\Lambda) = \|\mathbf{x}\|\|\mathbf{x}'\| \cdot \rho(r), \quad \text{and}$$
$$\tilde{\phi}'(\Lambda) = \rho'(r)$$

Definition 3.2. *For any function f, define the notation $f^m(x) = \underbrace{(f \circ \cdots \circ f)}_{m \text{ times}}(x)$.*

Definition 3.2 can be thought of as recursively applying function f to the input m times. For our dual activation function, we can show that repeating the function input will still result in a non-decreasing function.

Lemma 3.2. *For any $n \in \mathbb{N}$, $\rho^n(r)$ is a non-decreasing function.*

Proof. We use induction on n. For the base case of $n = 1$,

$$\frac{d\rho}{dr}(r) = 1 - \frac{\cos^{-1} r}{\pi} \geq 0.$$

Moreover, from Equation 3.6 we have that $\rho(-1) = 0$ and $\rho(1) = 1$, implying that

$$\rho(r) \in [0, 1] \subset [-1, 1], \quad \forall r \in [-1, 1],$$

and hence, $\rho^n(r) \in [-1, 1], \ \forall n \in \mathbb{N}$.

Now, assume that ρ^n is a non-decreasing function. Consider $r, s \in [-1, 1]$. If $r \leq s$, then $\rho^n(r) \leq \rho^n(s)$. Since we know $\rho^n(r), \rho^n(s) \in [-1, 1]$, we can conclude that $\rho^{n+1}(r) \leq \rho^{n+1}(s)$. □

Similarly, we can show that the dual of ϕ' is non-decreasing.

Lemma 3.3. *ρ' is non-decreasing with respect to r.*

Proof. Differentiating Equation 3.7 we get

$$\frac{d\rho'}{dr} = \frac{1}{\pi\sqrt{1 - r^2}} \geq 0.$$

Hence, ρ' is an increasing function. □

3.2.5 Bounding $\mathcal{K}$ with ρ

We will first establish a relationship between $\mathcal{K}$ and ρ.

Lemma 3.4. *For $l \in \{1, \ldots, L+1\}$,*

$$\mathcal{K}^{(l)}(\mathbf{x}, \mathbf{x}') = \sqrt{u^{(l-1)}} \cdot \rho\left(r_u^{(l)}\right) + \sigma_b^2$$

where $u^{(l)} = \left(\|\mathbf{x}\|^2 + l\sigma_b^2\right)\left(\|\mathbf{x}'\|^2 + l\sigma_b^2\right)$ and $r_u^{(l)} = \frac{\mathcal{K}^{(l-1)}(\mathbf{x}, \mathbf{x}')}{\sqrt{u^{(l-1)}}}$

Proof. We use induction on l. For the base case of $l = 1$,

$$\begin{aligned}\mathcal{K}^{(1)}(\mathbf{x}, \mathbf{x}') &= \tilde{\phi}\left(\begin{bmatrix}\mathcal{K}^{(0)}(\mathbf{x}, \mathbf{x}) & \mathcal{K}^{(0)}(\mathbf{x}, \mathbf{x}') \\ \mathcal{K}^{(0)}(\mathbf{x}', \mathbf{x}) & \mathcal{K}^{(0)}(\mathbf{x}', \mathbf{x}')\end{bmatrix}\right) + \sigma_b^2 \\ &= \tilde{\phi}\left(\begin{bmatrix}\mathbf{x}^T\mathbf{x} & \mathbf{x}^T\mathbf{x}' \\ \mathbf{x}'^T\mathbf{x} & \mathbf{x}'^T\mathbf{x}'\end{bmatrix}\right) + \sigma_b^2 \\ &= \|\mathbf{x}\|\|\mathbf{x}'\| \cdot \rho\left(\frac{\mathbf{x}^T\mathbf{x}'}{\|\mathbf{x}\|\|\mathbf{x}'\|}\right) + \sigma_b^2\end{aligned}$$

Now, assume that $\mathcal{K}^{(l)}(\mathbf{x}, \mathbf{x}') = \sqrt{u^{(l-1)}} \cdot \rho\left(r_u^{(l)}\right) + \sigma_b^2$ holds. Then,

$$\begin{aligned}\mathcal{K}^{(l+1)}(\mathbf{x}, \mathbf{x}') &= \tilde{\phi}\left(\begin{bmatrix}\mathcal{K}^{(l)}(\mathbf{x}, \mathbf{x}) & \mathcal{K}^{(l)}(\mathbf{x}, \mathbf{x}') \\ \mathcal{K}^{(l)}(\mathbf{x}', \mathbf{x}) & \mathcal{K}^{(l)}(\mathbf{x}', \mathbf{x}')\end{bmatrix}\right) + \sigma_b^2 \\ &= \tilde{\phi}\left(\begin{bmatrix}(\|\mathbf{x}\|^2 + (l-1)\sigma_b^2)\rho(1) + \sigma_b^2 & \mathcal{K}^{(l)}(\mathbf{x}, \mathbf{x}') \\ \mathcal{K}^{(l)}(\mathbf{x}', \mathbf{x}) & (\|\mathbf{x}'\|^2 + (l-1)\sigma_b^2)\rho(1) + \sigma_b^2\end{bmatrix}\right) + \sigma_b^2 \\ &= \tilde{\phi}\left(\begin{bmatrix}\|\mathbf{x}\|^2 + l\sigma_b^2 & \mathcal{K}^{(l)}(\mathbf{x}, \mathbf{x}') \\ \mathcal{K}^{(l)}(\mathbf{x}', \mathbf{x}) & \|x'\|^2 + l\sigma_b^2\end{bmatrix}\right) + \sigma_b^2 \\ &= \sqrt{u^{(l)}} \cdot \rho\left(\frac{\mathcal{K}^{(l)}(\mathbf{x}, \mathbf{x}')}{\sqrt{u^{(l)}}}\right) + \sigma_b^2\end{aligned}$$

where we use the fact that $\mathcal{K}^{(l)}(\mathbf{x}, \mathbf{x}') \leq \sqrt{\mathcal{K}^{(l)}(\mathbf{x}, \mathbf{x}) \cdot \mathcal{K}^{(l)}(\mathbf{x}', \mathbf{x}')}$. This proves the inductive step. □

Next, we will bound $\mathcal{K}^{(l)}(\mathbf{x}, \mathbf{x}')$ using ρ.

Lemma 3.5.

$$\tilde{\rho}_-^{(l)} \leq \mathcal{K}^{(l)}(\mathbf{x}, \mathbf{x}') \leq \tilde{\rho}_+^{(l)}$$

where

$$\tilde{\rho}_\pm^{(l)} = \begin{cases}\pm\|\mathbf{x}\|\|\mathbf{x}'\|, & \text{if} \quad l = 0, \\ \sqrt{u^{(l-1)}} \cdot \rho\left(\frac{\tilde{\rho}_\pm^{(l-1)}}{\sqrt{u^{(l-1)}}}\right) + \sigma_b^2, & \text{if} \quad l \geq 1\end{cases}$$

Proof. We use induction on l. For the base case of $l = 1$,

$$\mathcal{K}^{(1)}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x}\|\|\mathbf{x}'\| \cdot \rho\left(\frac{\mathbf{x}^T\mathbf{x}'}{\|\mathbf{x}\|\|\mathbf{x}'\|}\right) + \sigma_b^2$$

as in Lemma 3.4. Moreover, we have that

$$\tilde{\rho}_\pm^{(1)} = \|\mathbf{x}\|\|\mathbf{x}'\| \cdot \rho(\pm 1) + \sigma_b^2$$

We can simply conclude the result for $l = 1$ since ρ is non-decreasing.

Now, assume that $\tilde{\rho}_-^{(l)} \leq \mathcal{K}^{(l)}(\mathbf{x}, \mathbf{x}') \leq \tilde{\rho}_+^{(l)}$. Then,

$$\begin{aligned}
\mathcal{K}^{(l+1)}(\mathbf{x}, \mathbf{x}') &= \sqrt{u^{(l)}} \cdot \rho\left(\frac{\mathcal{K}^{(l)}(\mathbf{x}, \mathbf{x}')}{\sqrt{u^{(l)}}}\right) + \sigma_b^2 \\
&\leq \sqrt{u^{(l)}} \cdot \rho\left(\frac{\tilde{\rho}_+^{(l)}}{\sqrt{u^{(l)}}}\right) + \sigma_b^2 \\
&= \tilde{\rho}_+
\end{aligned}$$

A similar logic can be used to show that $\tilde{\rho}_- \leq \mathcal{K}^{(l+1)}(\mathbf{x}, \mathbf{x}')$. This proves the inductive step. □

Corollary 3.1. *For all $l \in \{1, \ldots, L+1\}$,*

$$\sqrt{u^{(l-1)}} \cdot \rho\left(\hat{r}_-^{(l)}\right) + \sigma_b^2 \leq \mathcal{K}^{(l)}(\mathbf{x}, \mathbf{x}') \leq \sqrt{u^{(l-1)}} \cdot \rho\left(\hat{r}_+^{(l)}\right) + \sigma_b^2$$

where

$$\begin{aligned}
&\hat{r}_+^{(l)} = 1, \quad \textit{and} \\
&\hat{r}_-^{(l)} = \begin{cases} -1, & \textit{if} \quad l = 1, \\ \rho\left(\hat{r}_-^{(l-1)}\right) \cdot \frac{l-2}{l-1}, & \textit{if} \quad l > 1 \end{cases}
\end{aligned}$$

Proof. Notice that by Lemma 3.5 and due to the non-decreasing nature of ρ, proving the corollary above is equivalent to showing that $\frac{\tilde{\rho}_+^{(l-1)}}{\sqrt{u^{(l-1)}}} \leq r_+^{(l)}$ and $\frac{\tilde{\rho}_-^{(l-1)}}{\sqrt{u^{(l-1)}}} \geq r_-^{(l)}$.

We will start by showing that $\frac{\tilde{\rho}_+^{(l-1)}}{\sqrt{u^{(l-1)}}} \leq r_+^{(l)} = 1$. For the base case of $l = 1$, the result is trivial since

$$\frac{\tilde{\rho}_+^{(0)}}{\sqrt{u^{(0)}}} = 1$$

Now, assume that $\frac{\tilde{\rho}_+^{(l-1)}}{\sqrt{u^{(l-1)}}} \leq 1$. Since $0 \leq \rho(r) \leq 1, \forall r \in [-1, 1]$, using Lemma 3.4, we have

$$\begin{aligned}
\left(\frac{\tilde{\rho}_+^{(l-1)}}{\sqrt{u^{(l-1)}}}\right)^2 &\leq \left(\frac{\sqrt{u^{(l-2)}} + \sigma_b^2}{\sqrt{u^{(l-1)}}}\right)^2 \\
&= \frac{\sigma_b^4 + t(\mathbf{x})\, t(\mathbf{x}') + \sigma_b^2 \left(2\sqrt{t(\mathbf{x})\, t(\mathbf{x}')}\right)}{\sigma_b^4 + t(\mathbf{x})\, t(\mathbf{x}') + \sigma_b^2 (t(\mathbf{x}) + t(\mathbf{x}'))} \\
&\leq 1,
\end{aligned}$$

where $t(\cdot) = (\|\cdot\| + (l-2)\sigma_b^2)$. Therefore, $\frac{\hat{\rho}_+^{(l-1)}}{\sqrt{u^{(l-1)}}} \leq 1$. This proves the first part of the corollary.

For the second part, we will use induction on l. It is again trivial to show that the statement

is true for $l = 1$. Now, assume that $\frac{\tilde{\rho}_-^{(l-1)}}{\sqrt{u^{(l-1)}}} \geq r_-^{(l)}$. Then,

$$\begin{aligned}
\frac{\tilde{\rho}_-^{(l)}}{\sqrt{u^{(l)}}} &= \frac{\sqrt{u^{(l-1)}} \cdot \rho\left(\frac{\tilde{\rho}_-^{(l-1)}}{\sqrt{u^{(l-1)}}}\right) + \sigma_b^2}{\sqrt{u^{(l)}}} \\
&\geq \frac{\sqrt{u^{(l-1)}} \cdot \rho\left(r_-^{(l)}\right) + \sigma_b^2}{\sqrt{u^{(l)}}} \\
&\geq \rho\left(r_-^{(l)}\right) \cdot \frac{\sqrt{u^{(l-1)}} + \sigma_b^2}{\sqrt{u^{(l)}}} \\
&\geq \rho\left(r_-^{(l)}\right) \cdot \frac{l-1}{l} \\
&= r_-^{(l+1)}
\end{aligned}$$

where in we use the following result:

$$\frac{\sqrt{u^{(l-1)}} + \sigma_b^2}{\sqrt{u^{(l)}}} \geq \frac{\sqrt{u^{(l-1)}}}{\sqrt{u^{(l)}}} \geq \sqrt{\frac{\|\mathbf{x}\| + (l-1)\,\sigma_b^2}{\|\mathbf{x}\| + (l)\,\sigma_b^2}} \cdot \sqrt{\frac{\|\mathbf{x}'\| + (l-1)\,\sigma_b^2}{\|\mathbf{x}'\| + (l)\,\sigma_b^2}} \geq \frac{l-1}{l} \tag{3.8}$$

This proves the second part of our corollary. □

3.2.6 Bounding $\dot{\mathcal{K}}$ with ρ

We will first show that we can express $\dot{\mathcal{K}}$ in terms of ρ'.

Lemma 3.6. *For $l \in \{1, \ldots, L+1\}$,*

$$\dot{\mathcal{K}}^{(l)}(\mathbf{x}, \mathbf{x}') = \rho'\left(r_u^{(l)}\right)$$

where $r_u^{(l)}$ is defined in Lemma 3.4.

Proof. We can show that

$$\begin{aligned}
\dot{\mathcal{K}}^{(l)}(\mathbf{x}, \mathbf{x}') &= \tilde{\phi}'\left(\begin{bmatrix} \mathcal{K}^{(l-1)}(\mathbf{x}, \mathbf{x}) & \mathcal{K}^{(l-1)}(\mathbf{x}, \mathbf{x}') \\ \mathcal{K}^{(l-1)}(\mathbf{x}', \mathbf{x}) & \mathcal{K}^{(l-1)}(\mathbf{x}', \mathbf{x}') \end{bmatrix}\right) \\
&= \tilde{\phi}'\left(\begin{bmatrix} (\|x\|^2 + (l-2)\,\sigma_b^2)\,\rho(1) + \sigma_b^2 & \mathcal{K}^{(l-1)}(\mathbf{x}, \mathbf{x}') \\ \mathcal{K}^{(l-1)}(\mathbf{x}', \mathbf{x}) & (\|x'\|^2 + (l-2)\,\sigma_b^2)\,\rho(1) + \sigma_b^2 \end{bmatrix}\right) \\
&= \tilde{\phi}'\left(\begin{bmatrix} \|x\|^2 + (l-1)\,\sigma_b^2 & \mathcal{K}^{(l-1)}(\mathbf{x}, \mathbf{x}') \\ \mathcal{K}^{(l-1)}(\mathbf{x}', \mathbf{x}) & \|x'\|^2 + (l-1)\,\sigma_b^2 \end{bmatrix}\right) \\
&= \rho'\left(\frac{\mathcal{K}^{(l-1)}(\mathbf{x}, \mathbf{x}')}{\sqrt{u^{(l-1)}}}\right)
\end{aligned}$$

□

Using the result above, we can now bound the values of $\dot{\mathcal{K}}^{(l)}(\mathbf{x}, \mathbf{x}')$.

Corollary 3.2.

$$\rho'\left(\hat{r}_-^{(l)}\right) \leq \dot{\mathcal{K}}^{(l)}\left(\mathbf{x}, \mathbf{x}'\right) \leq \rho'\left(\hat{r}_+^{(l)}\right)$$

where $\hat{r}_-^{(l)}$ *and* $\hat{r}_+^{(l)}$ *are as defined in Corollary 3.1.*

Proof. From Corollary 3.1, we know that $\hat{r}_-^{(l)} \leq r_u^{(l)} \leq \hat{r}_+^{(l)}$. The result follows from Lemma 3.3. □

3.2.7 Ratio between $\mathcal{K}$ and Θ

We will now prove the bound on the ratio between $\mathcal{K}$ and Θ.

Theorem 3.1. *For a neural network with* $L \geq 2$*, if* $\max\{\|x\|, \|x'\|\} \leq B$*, then*

$$1 + \sum_{l=1}^{L} \rho\left(\hat{r}_-^{(l)}\right) \cdot \frac{l-1}{L+1} \prod_{l'=l+1}^{L+1} \rho'\left(\hat{r}_-^{(l')}\right) \leq \frac{\Theta\left(\mathbf{x}, \mathbf{x}'\right)}{\mathcal{K}\left(\mathbf{x}, \mathbf{x}'\right)} \leq 1 + \frac{L}{\rho\left(\hat{r}_-^{(L+1)}\right)}$$

Proof. We will first prove the right hand inequality. We see that

$$\begin{aligned}
\frac{\Theta\left(\mathbf{x}, \mathbf{x}'\right)}{\mathcal{K}\left(\mathbf{x}, \mathbf{x}'\right)} &= \frac{1}{\mathcal{K}^{(L+1)}\left(\mathbf{x}, \mathbf{x}'\right)} \cdot \sum_{l=1}^{L+1} \mathcal{K}^{(l)}\left(\mathbf{x}, \mathbf{x}'\right) \prod_{l'=l+1}^{L+1} \dot{\mathcal{K}}^{(l')}\left(\mathbf{x}, \mathbf{x}'\right) \\
&\leq 1 + \sum_{l=1}^{L} \frac{\sqrt{u^{(l-1)}} \cdot \rho\left(\hat{r}_+^{(l)}\right) + \sigma_b^2}{\sqrt{u^{(l)}} \cdot \rho\left(\hat{r}_-^{(L+1)}\right) + \sigma_b^2} \prod_{l'=l+1}^{L+1} \rho'\left(\hat{r}_+^{(l')}\right) \\
&= 1 + \sum_{l=1}^{L} \frac{1}{\rho\left(\hat{r}_-^{(L+1)}\right)} \cdot \frac{\sqrt{u^{(l-1)}} + \sigma_b^2}{\sqrt{u^{(l)}}} \\
&\leq 1 + \frac{L}{\rho\left(\hat{r}_-^{(L+1)}\right)}
\end{aligned}$$

where we use the fact that $\frac{\sqrt{u^{(l-1)}} + \sigma_b^2}{\sqrt{u^{(l)}}} \leq 1$ based on a similar argument used in Equation 3.8.

Similarly for the left hand inequality,

$$\begin{aligned}
\frac{\Theta\left(\mathbf{x}, \mathbf{x}'\right)}{\mathcal{K}\left(\mathbf{x}, \mathbf{x}'\right)} &\geq 1 + \sum_{l=1}^{L} \frac{\sqrt{u^{(l-1)}} \cdot \rho\left(\hat{r}_-^{(l)}\right) + \sigma_b^2}{\sqrt{u^{(l)}} \cdot \rho\left(\hat{r}_+^{(L+1)}\right) + \sigma_b^2} \prod_{l'=l+1}^{L+1} \rho'\left(\hat{r}_-^{(l')}\right) \\
&\geq 1 + \sum_{l=1}^{L} \rho\left(\hat{r}_-^{(l)}\right) \cdot \frac{\sqrt{u^{(l-1)}}}{\sqrt{u^{(l)}} + \sigma_b^2} \prod_{l'=l+1}^{L+1} \rho'\left(\hat{r}_-^{(l')}\right) \\
&\geq 1 + \sum_{l=1}^{L} \rho\left(\hat{r}_-^{(l)}\right) \cdot \frac{l-1}{L+1} \prod_{l'=l+1}^{L+1} \rho'\left(\hat{r}_-^{(l')}\right),
\end{aligned}$$

where we use the fact that $\frac{\sqrt{u^{(l-1)}}}{\sqrt{u^{(l)}}+\sigma_b^2} \geq \frac{\sqrt{u^{(l-1)}}}{\sqrt{u^{(L+1)}}} \geq \frac{l-1}{L+1}$ based on a similar line of reasoning as in Equation 3.8. This proves our theorem. □

From Theorem 3.1, we are able to give a bound for the ratio between Θ and $\mathcal{K}$. While the constant is defined recursively based on the function ρ, we claim that this is still useful since it is expressed in a form which allows it to be computed directly, and more importantly, we are able to see that such a constant exists.

In the case that there is no bias, we can improve the bound on the ratio between $\mathcal{K}$ and Θ. The key fact is that in the case without bias, $u^{(l)}$ is equal for all values of l, and we can simplify $\hat{\rho}_{\pm}^{(l)} = \|\mathbf{x}\|\|\mathbf{x}'\| \cdot \rho^l(\pm 1)$. During the computation, the constant $\|\mathbf{x}\|\|\mathbf{x}'\|$ will then cancel out in several places.

Theorem 3.2. *For a neural network with $L \geq 2$ and $\sigma_b = 0$,*

$$\sum_{l=1}^{L+1} \frac{\rho^l(-1)}{\rho^{L+1}(1)} \prod_{l'=l+1}^{L+1} \left(\rho' \circ \rho^{l'}\right)(-1) \leq \frac{\Theta(\mathbf{x}, \mathbf{x}')}{\mathcal{K}(\mathbf{x}, \mathbf{x}')} \leq \sum_{l=1}^{L+1} \frac{\rho^l(1)}{\rho^{L+1}(-1)} \prod_{l'=l+1}^{L+1} \left(\rho' \circ \rho^{l'}\right)(1)$$

Proof. It is easy to see from the recursive form in Equation 3.2 that if $\sigma_b = 0$, then $\mathcal{K}^{(l)}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x}\|\|\mathbf{x}'\| \cdot \rho^l\left(\frac{\mathbf{x}^T\mathbf{x}'}{\|\mathbf{x}\|\|\mathbf{x}'\|}\right)$. Since we know ρ^l is non-decreasing from Lemma 3.2, we are able to bound

$$\|\mathbf{x}\|\|\mathbf{x}'\| \cdot \rho^l(-1) \leq \mathcal{K}^{(l)}(\mathbf{x}, \mathbf{x}') \leq \|\mathbf{x}\|\|\mathbf{x}'\| \cdot \rho^l(1) \tag{3.9}$$

Similarly, we can also see from Equation 3.4 that $\dot{\mathcal{K}}^{(l)}(\mathbf{x}, \mathbf{x}') = (\rho' \circ \rho^l)\left(\frac{\mathbf{x}^T\mathbf{x}'}{\|\mathbf{x}\|\|\mathbf{x}'\|}\right)$. Since ρ' is non-decreasing based on Lemma 3.3, we are able to bound

$$(\rho' \circ \rho^l)(-1) \leq \dot{\mathcal{K}}^{(l)}(\mathbf{x}, \mathbf{x}') \leq (\rho' \circ \rho^l)(1) \tag{3.10}$$

Given that we can write $\Theta(\mathbf{x}, \mathbf{x}')$ with the recursive form given by Equation 3.5, it is then simple to use Equation 3.9 and Equation 3.10 to bound $\frac{\Theta(\mathbf{x}, \mathbf{x}')}{\mathcal{K}(\mathbf{x}, \mathbf{x}')}$. □

3.2.8 Bounding the Difference Between $\sigma_{\mathbf{NN}}$ and $\sigma_{\mathbf{NTKGP}}$

From above, we are able to see that it is possible to bound the ratio

$$a_- \leq \frac{\mathcal{K}(\mathbf{x}, \mathbf{x}')}{\Theta(\mathbf{x}, \mathbf{x}')} \leq a_+ \tag{3.11}$$

for some appropriately set a_- and a_+ according to either Theorem 3.1 or Theorem 3.2 depending on the value of σ_b (note that in the two theorems above we show the bounds for the reciprocal of what is stated in Equation 3.11). Given this bound, we are able to show the following main result.

Theorem 3.3. *For a neural network with ReLU activation and $L \geq 2$ hidden layers, if $\Theta_{\mathbf{X}} \succeq 0$, then*

$$\left|\sigma_{NN}^2(\mathbf{x}|\mathcal{D}) - \alpha \cdot \sigma_{NTKGP}^2(\mathbf{x}|\mathcal{D})\right| \leq \beta$$

where n is the upper limit on the size of the training set, $B \geq |\Theta(\mathbf{x}, \mathbf{x}')|$, $\alpha \in [a_-, a_+]$, $\gamma = B \cdot \max\{\alpha - a_-, a_+ - \alpha\}$, and $\beta = \gamma + \frac{n\gamma B^2}{\lambda_{min}(\Theta_{\mathbf{X}})^2} + \frac{2n\gamma B}{\lambda_{min}(\Theta_{\mathbf{X}})}$.

Proof. First, we know that we have $\frac{\mathcal{K}(\mathbf{x},\mathbf{x}')}{\Theta(\mathbf{x},\mathbf{x}')} \in [a_-, a_+]$ by assumption. In the case that $\Theta(\mathbf{x}, \mathbf{x}') \geq 0$, we can convert the multiplicative bound into an additive bound as

$$\begin{aligned} \mathcal{K}(\mathbf{x}, \mathbf{x}') &\geq a_- \cdot \Theta(\mathbf{x}, \mathbf{x}') \\ \alpha \cdot \Theta(\mathbf{x}, \mathbf{x}') - \mathcal{K}(\mathbf{x}, \mathbf{x}') &\leq (\alpha - a_-) \cdot \Theta(\mathbf{x}, \mathbf{x}') \\ &\leq (\alpha - a_-) \cdot B \end{aligned}$$

and

$$\begin{aligned} \mathcal{K}(\mathbf{x}, \mathbf{x}') &\leq a_+ \Theta(\mathbf{x}, \mathbf{x}') \\ \mathcal{K}(\mathbf{x}, \mathbf{x}') - \alpha \cdot \Theta(\mathbf{x}, \mathbf{x}') &\leq (\alpha - a_+) \cdot \Theta(\mathbf{x}, \mathbf{x}') \\ &\leq (\alpha - a_+) \cdot B \end{aligned}$$

which can be combined to give $|\mathcal{K}(\mathbf{x}, \mathbf{x}') - \alpha \cdot \Theta(\mathbf{x}, \mathbf{x}')| \leq \gamma$ for γ as defined earlier. The same is the case when $\Theta(\mathbf{x}, \mathbf{x}') < 0$.

Given this additive bound, we are then able to bound each term which appears in σ_{NN} individually. We can see that

$$\begin{aligned} \left|\mathcal{K}_{\mathbf{xX}}\Theta_{\mathbf{X}}^{-1}\Theta_{\mathbf{Xx}} - \alpha \cdot \Theta_{x\mathbf{X}}\Theta_{\mathbf{X}}^{-1}\Theta_{\mathbf{Xx}}\right| &= \left|\left(\mathcal{K}_{\mathbf{xX}} - \alpha \cdot \Theta_{\mathbf{xX}}\right)\Theta_{\mathbf{X}}^{-1}\Theta_{\mathbf{Xx}}\right| \\ &\leq \lambda_{\max}\left(\Theta_{\mathbf{X}}^{-1}\right)\|\mathcal{K}_{\mathbf{Xx}} - \alpha \cdot \Theta_{\mathbf{Xx}}\|\|\Theta_{\mathbf{Xx}}\| \\ &\leq \frac{1}{\lambda_{\min}(\Theta_{\mathbf{X}})} \cdot \gamma\sqrt{n} \cdot B\sqrt{n} \\ &= \frac{n\gamma B}{\lambda_{\min}(\Theta_{\mathbf{X}})} \end{aligned}$$

Similarly,

$$\begin{aligned} \left|\Theta_{\mathbf{xX}}\Theta_{\mathbf{X}}^{-1}\mathcal{K}_{\mathbf{X}}\Theta_{\mathbf{X}}^{-1}\Theta_{\mathbf{Xx}} - \alpha \cdot \Theta_{\mathbf{xX}}\Theta_{\mathbf{X}}^{-1}\Theta_{\mathbf{Xx}}\right| &= \left|\Theta_{\mathbf{xX}}\Theta_{\mathbf{X}}^{-1}\left(\mathcal{K}_{\mathbf{X}} - \alpha \cdot \Theta_{\mathbf{x}}\right)\Theta_{\mathbf{X}}^{-1}\Theta_{\mathbf{Xx}}\right| \\ &\leq \lambda_{\max}\left(\mathcal{K}_{\mathbf{X}} - \alpha \cdot \Theta_{\mathbf{X}}\right)\|\Theta_{\mathbf{X}}^{-1}\Theta_{\mathbf{Xx}}\|^2 \\ &= \lambda_{\max}\left(\mathcal{K}_{\mathbf{X}} - \alpha \cdot \Theta_{\mathbf{X}}\right) \cdot \lambda_{\max}\left(\Theta_{\mathbf{X}}^{-1}\right)^2 \cdot \|\Theta_{\mathbf{X}x}\|^2 \\ &\leq \frac{n\gamma B^2}{\lambda_{\min}(\Theta_{\mathbf{X}})^2} \end{aligned}$$

Combining these results together, we obtain

$$\begin{aligned}
\left|\sigma^2_{\text{NN}}\left(\mathbf{x}|\mathcal{D}\right) - \alpha \cdot \sigma^2_{\text{NTKGP}}\left(\mathbf{x}|\mathcal{D}\right)\right| &= \left|\mathcal{K}_{\mathbf{x}} + \Theta_{\mathbf{x}\mathbf{X}}\Theta_{\mathbf{X}}^{-1}\mathcal{K}_{\mathbf{X}}\Theta_{\mathbf{X}}^{-1}\Theta_{\mathbf{X}\mathbf{x}} - \mathcal{K}_{\mathbf{x}\mathbf{X}}\Theta_{\mathbf{X}}^{-1}\Theta_{\mathbf{X}\mathbf{x}}\right. \\
&\quad \left. - \Theta_{\mathbf{x}\mathbf{X}}\Theta_{\mathbf{X}}^{-1}\mathcal{K}_{\mathbf{X}\mathbf{x}} - \alpha\left(\Theta_{\mathbf{x}} - \Theta_{\mathbf{x}\mathbf{X}}\Theta_{\mathbf{X}}^{-1}\Theta_{\mathbf{X}\mathbf{x}}\right)\right| \\
&\leq \left|\mathcal{K}_{\mathbf{x}} - \alpha\Theta_{\mathbf{x}}\right| \\
&\quad + \left|\Theta_{\mathbf{x}\mathbf{X}}\Theta_{\mathbf{X}}^{-1}\mathcal{K}_{\mathbf{X}}\Theta_{\mathbf{X}}^{-1}\Theta_{\mathbf{X}\mathbf{x}} - \alpha\Theta_{\mathbf{x}\mathbf{X}}\Theta_{\mathbf{X}}^{-1}\Theta_{\mathbf{X}\mathbf{x}}\right| \\
&\quad + \left|\mathcal{K}_{\mathbf{x}\mathbf{X}}\Theta_{\mathbf{X}}^{-1}\Theta_{\mathbf{X}\mathbf{x}} - \alpha\Theta_{\mathbf{x}\mathbf{X}}\Theta_{\mathbf{X}}^{-1}\Theta_{\mathbf{X}\mathbf{x}}\right| \\
&\quad + \left|\Theta_{\mathbf{x}\mathbf{X}}\Theta_{\mathbf{X}}^{-1}\mathcal{K}_{\mathbf{X}\mathbf{x}} - \alpha\Theta_{\mathbf{x}\mathbf{X}}\Theta_{\mathbf{X}}^{-1}\Theta_{\mathbf{X}\mathbf{x}}\right| \\
&\leq \gamma + \frac{n\gamma B^2}{\lambda_{\min}\left(\Theta_{\mathbf{X}}\right)^2} + \frac{2n\gamma B}{\lambda_{\min}\left(\Theta_{\mathbf{X}}\right)}
\end{aligned}$$

□

3.3 Connection with Generalization Error

In this section, we show that the approximate output variance, σ^2_{NTKGP}, is also an upper bound on the generalization error of the trained neural network, and hence, a good indicator of its predictive performance.

To analyze the performance of the trained neural network, we make the following assumption about the groundtruth function f^* [68].

Assumption 3.1. *Assume that the groundtruth function, f^*, lies in the reproducing kernel Hilbert space RKHS, $\mathcal{H}_\Theta$, of the NTK Θ, or equivalently, its RKHS norm satisfies $\|f^*\|_{\mathcal{H}_\Theta} \leq B$ for some $B \in \mathbb{R}_{\geq 0}$.*

Assumption 3.2. *Assume that the function observation at any input $\mathbf{x}_i$ is given by $y_i = f^*\left(\mathbf{x}_i\right) + \xi_i$, in which every ξ_i is i.i.d. observation noise drawn from an R sub-Gaussian distribution: $\mathbb{E}\left[\exp\left(\eta\xi_i\right)\right] \leq \exp\left(\eta^2 R^2/2\right)$, $\forall \eta \in \mathbb{R}$.*

Both the assumptions, Assumption 3.1 and Assumption 3.2, are commonly made in the analysis of kernelized and neural bandits [10, 27]. They allow us to show the following theoretical guarantee on the generalization error:

Theorem 3.4. *Let the function $f^* \in \mathcal{H}_\Theta$ and training set $\mathcal{D} = (\mathbf{X}, \mathbf{y})$ follow Assumption 3.1 and Assumption 3.2. Suppose we train an infinitely wide neural network $f\left(\cdot;\theta\right)$ with mean-squared error loss function as given by Equation 2.9, using gradient descent until convergence. Then, for any $\mathbf{x} \in \mathcal{X}$, with probability at least $1 - 2\delta$ over the random observation noise and network initialization,*

$$\left|f^*\left(\mathbf{x}\right) - f\left(\mathbf{x}; \mathit{train}\left(\theta_0\right)\right)\right| \leq \left[B + \left(\frac{R}{\lambda} + 1\right)\sqrt{2\log\delta^{-1}}\right]\sigma_{\mathit{NTKGP}}\left(\mathbf{x}|\mathcal{D}\right),$$

where λ is the regularization of the loss function.

Proof. Using the triangle inequality, we can show that

$$
\begin{aligned}
& \left|f^*(\mathbf{x}) - f\left(\mathbf{x}; \text{train}\left(\theta_0\right)\right)\right| \\
\leq\ & \left|f^*(\mathbf{x}) - \mu\left(\mathbf{x}|\mathcal{D}\right)\right| + \left|\mu\left(\mathbf{x}|\mathcal{D}\right) - f\left(\mathbf{x}; \theta_\infty\right)\right| \\
\leq\ & \left(B + \frac{R}{\lambda}\sqrt{2\log\delta^{-1}}\right)\sigma_{\text{NTKGP}}\left(\mathbf{x}|\mathcal{D}\right) + \left|\mu\left(\mathbf{x}|\mathcal{D}\right) - f(\mathbf{x}; \theta_\infty)\right| \qquad (3.12) \\
\leq\ & \left(B + \frac{R}{\lambda}\sqrt{2\log\delta^{-1}}\right)\sigma_{\text{NTKGP}}\left(\mathbf{x}|\mathcal{D}\right) + \sqrt{2\log\delta^{-1}}\cdot\sigma_{\text{NN}}\left(\mathbf{x}|\mathcal{D}\right) \qquad (3.13) \\
\leq\ & \left[B + \left(\frac{R}{\lambda} + 1\right)\sqrt{2\log\delta^{-1}}\right]\sigma_{\text{NTKGP}}\left(\mathbf{x}|\mathcal{D}\right),
\end{aligned}
$$

where Equation 3.12 is true with probability at least $1-\delta$ using Theorem 1 of [67], and Equation 3.13 is true with probability at least $1-\delta$ due to Hoeffding inequality for sub-Gaussian random variables. By union bound, the statement above is true with probability at least $1-2\delta$. □

Theorem 3.4 shows that the generalization error of a neural network trained using gradient descent is proportional to σ_{NTKGP}, where the constant of proportionality, ζ, is independent of $\mathbf{x}$ and $\mathcal{D}$. As a result of Theorem 3.4, we know that minimizing σ_{NTKGP} will not only (a) decrease the output variance, and hence improve initialization robustness, but also (b) reduce the generalization error and hence enhance the predictive performance.

The degree of correlation between σ_{NTKGP} and the generalization performance, represented by the constant ζ, depends on the parameters B and R, such that easier the function f^* is to learn (i.e., a smaller B) or lesser noisy the observations are (i.e., a smaller R), the better the degree of correlation.

Theorem 3.4 is also consistent with the empirically observed characteristics of over-parameterised neural networks, because neural network models with lower variance are also observed to have higher predictive accuracy [43].

3.4 AL Criterion

Since we have shown that minimizing $\sigma_{\text{NTKGP}}\left(\mathbf{x}|\mathcal{D}\right)$ can improve both initialization robustness (Section 3.2) and generaliztion performance (Section 3.3), we design our AL criterion based on the minimization of $\sigma_{\text{NTKGP}}\left(\mathbf{x}|\mathcal{D}\right)$ across all test input points $\mathbf{x} \in \mathbf{X}_T$. Specifically, our EV-GP criterion encourages the selection of input data points which result in small expected output variance $\sigma^2_{\text{NTKGP}}\left(\mathbf{x}|\mathcal{D}\right)$ across all test inputs, and we estimate the expected variance by averaging $\sigma^2_{\text{NTKGP}}\left(\mathbf{x}|\mathcal{D}\right)$ over the available test set $\mathbf{X}_T$:

$$
\alpha_{\text{EV}}\left(\mathbf{X}\right) = \frac{1}{|\mathbf{X}_T|}\sum_{\mathbf{x}\in\mathbf{X}_T}\left[\sigma^2_{\text{NTKGP}}\left(\mathbf{x}|\phi\right) - \sigma^2_{\text{NTKGP}}\left(\mathbf{x}|\mathcal{D}\right)\right]. \qquad (3.14)
$$

We have added $\sigma^2_{\text{NTKGP}}(\mathbf{x}|\phi)$ to the criterion so that $\alpha_{\text{EV}}(\mathbf{X}) \geq 0$ and that our criterion is to be maximized during AL. The EV-GP criterion has multiple computational benefits:

1. Firstly, it is training-free, i.e., its calculation does not require any training of the neural network and is hence able to sidestep significant computational costs resulting from model training.

2. Secondly, it only requires calculating the variance at individual test points rather than the full covariance over the testing set.

3. Thirdly, it can make use of the approximation techniques based on sparse GPs discussed in Subsection 2.3.2, for which we simply need to replace σ^2_{NTKGP} (Equation 2.12) by its sparse GP counterparts in Equation 3.14.

4. Fourthly, it is monotone submodular, and therefore a greedy approach (i.e., in each round, select the point that maximizes the criterion) is guaranteed to give a $(1 - \frac{1}{e})$-optimal solution [46].

5. Finally, it is label-independent because the calculation of σ^2_{NTKGP} does not require the test observations, and therefore, does not need the heuristic of pseudo-labels unlike previous AL algorithms [4, 42].

As noted in Section 3.1, even with the NTKGP approximation, the covariance matrix computation is $\mathcal{O}(n^2)$. To make our algorithm suitable for practical settings, we propose strategies to further reduce the cost of using the EV-GP criterion.

3.4.1 Bounding the Ratio Between $\alpha_{\text{EV,NN}}$ and $\alpha_{\text{EV,NTKGP}}$

It turns out that we are able to get a bound on the ratio of the EV criterion directly when we use σ_{NN} versus when we use σ_{NTKGP}. Considering the case of a single test point, we can see that using σ_{NN}, the EV criterion gives

$$\alpha_{\text{EV,NN}}(\mathbf{x}|\mathcal{D}) = \sigma^2_{\text{NN}}(\mathbf{x}|\phi) - \sigma^2_{\text{NN}}(\mathbf{x}|\mathcal{D}) = 2 \cdot \Theta_{\mathbf{xX}}\Theta^{-1}_{\mathbf{X}}\mathcal{K}_{\mathbf{Xx}} - \Theta_{\mathbf{xX}}\Theta^{-1}_{\mathbf{X}}\mathcal{K}_{\mathbf{X}}\Theta^{-1}_{\mathbf{X}}\Theta_{\mathbf{Xx}} \geq 0,$$

and using σ_{NTKGP},

$$\alpha_{\text{EV,NTKGP}}(\mathbf{x}|\mathcal{D}) = \sigma^2_{\text{NTKGP}}(\mathbf{x}|\phi) - \sigma^2_{\text{NTKGP}}(\mathbf{x}|\mathcal{D}) = \Theta_{\mathbf{xX}}\Theta^{-1}_{\mathbf{X}}\Theta_{\mathbf{Xx}} \geq 0.$$

Based on this, we are able to show the following bound.

Lemma 3.7. *Assume* $\mathbf{X}$ *is such that* $\Theta_{\mathbf{X}}, \mathcal{K}_{\mathbf{X}} \succeq 0$. *Let* $\|\Theta_{\mathbf{X}}\|_\infty, \|\Theta_{\mathbf{Xx}}\|_\infty \leq B$ *and* $a_- \leq \frac{\mathcal{K}(\mathbf{x},\mathbf{x}')}{\Theta(\mathbf{x},\mathbf{x}')} \leq a_+$. *Then,*

$$\frac{2a_- \cdot \lambda_{min}(\Theta_{\mathbf{X}})}{B} - \frac{a_+ B}{\lambda_{min}(\Theta_{\mathbf{X}})} \leq \frac{\alpha_{EV,NN}(\mathbf{x}|\mathcal{D})}{\alpha_{EV,NTKGP}(\mathbf{x}|\mathcal{D})} \leq \frac{2a_+ B}{\lambda_{min}(\Theta_{\mathbf{X}})}.$$

Proof. For the right hand inequality, we can see that

$$\begin{aligned}\frac{\alpha_{\text{EV,NN}}\left(\mathbf{x}|\mathcal{D}\right)}{\alpha_{\text{EV,NTKGP}}\left(\mathbf{x}|\mathcal{D}\right)} &\leq \frac{2\cdot\Theta_{\mathbf{x}\mathbf{X}}\Theta_{\mathbf{X}}^{-1}\mathcal{K}_{\mathbf{X}\mathbf{x}}}{\Theta_{\mathbf{x}\mathbf{X}}\Theta_{\mathbf{X}}^{-1}\Theta_{\mathbf{X}\mathbf{x}}} \\ &\leq \frac{2\cdot\lambda_{\max}\left(\Theta_{\mathbf{X}}^{-1}\right)\left\|\Theta_{\mathbf{X}\mathbf{x}}\right\|\left\|\mathcal{K}_{\mathbf{X}\mathbf{x}}\right\|}{\lambda_{\min}\left(\Theta_{\mathbf{X}}^{-1}\right)\left\|\Theta_{\mathbf{X}\mathbf{x}}\right\|^2} \\ &= \frac{2\cdot\lambda_{\max}\left(\Theta_{\mathbf{X}}\right)\left\|\mathcal{K}_{\mathbf{X}\mathbf{x}}\right\|}{\lambda_{\min}\left(\Theta_{\mathbf{X}}\right)\left\|\Theta_{\mathbf{X}\mathbf{x}}\right\|} \\ &\leq \frac{2a_+B}{\lambda_{\min}\left(\Theta_{\mathbf{X}}\right)}.\end{aligned}$$

Meanwhile, for the left hand inequality,

$$\begin{aligned}\frac{\alpha_{\text{EV,NN}}\left(\mathbf{x}|\mathcal{D}\right)}{\alpha_{\text{EV,NTKGP}}\left(\mathbf{x}|\mathcal{D}\right)} &= \frac{2\cdot\Theta_{\mathbf{x}\mathbf{X}}\Theta_{\mathbf{X}}^{-1}\mathcal{K}_{\mathbf{X}\mathbf{x}}}{\Theta_{\mathbf{x}\mathbf{X}}\Theta_{\mathbf{X}}^{-1}\Theta_{\mathbf{X}\mathbf{x}}} - \frac{\Theta_{\mathbf{x}\mathbf{X}}\Theta_{\mathbf{X}}^{-1}\mathcal{K}_{\mathbf{X}}\Theta_{\mathbf{X}}^{-1}\Theta_{\mathbf{X}\mathbf{x}}}{\Theta_{\mathbf{x}\mathbf{X}}\Theta_{\mathbf{X}}^{-1}\Theta_{\mathbf{X}\mathbf{x}}} \\ &\geq \frac{2\cdot\lambda_{\min}\left(\Theta_{\mathbf{X}}^{-1}\right)\left\|\Theta_{\mathbf{X}\mathbf{x}}\right\|\left\|\mathcal{K}_{\mathbf{X}\mathbf{x}}\right\|}{\lambda_{\max}\left(\Theta_{\mathbf{X}}^{-1}\right)\left\|\Theta_{\mathbf{X}\mathbf{x}}\right\|^2} - \frac{\lambda_{\max}\left(\mathcal{K}_{\mathbf{X}}\right)\left\|\Theta_{\mathbf{X}}^{-1}\Theta_{\mathbf{X}\mathbf{x}}\right\|^2}{\lambda_{\min}\left(\Theta_{\mathbf{X}}^{-1}\right)\left\|\Theta_{\mathbf{X}}^{-1}\Theta_{\mathbf{X}\mathbf{x}}\right\|^2} \\ &\geq \frac{2\cdot\lambda_{\min}\left(\Theta_{\mathbf{X}}^{-1}\right)\left\|\mathcal{K}_{\mathbf{X}\mathbf{x}}\right\|}{\lambda_{\max}\left(\Theta_{\mathbf{X}}^{-1}\right)\left\|\Theta_{\mathbf{X}\mathbf{x}}\right\|} - \frac{\lambda_{\max}\left(\mathcal{K}_{\mathbf{X}}\right)}{\lambda_{\min}\left(\Theta_{\mathbf{X}}\right)} \\ &\geq \frac{2\cdot\lambda_{\min}\left(\Theta_{\mathbf{X}}\right)\left\|\mathcal{K}_{\mathbf{X}\mathbf{x}}\right\|}{\lambda_{\max}\left(\Theta_{\mathbf{X}}\right)\left\|\Theta_{\mathbf{X}\mathbf{x}}\right\|} - \frac{\lambda_{\max}\left(\mathcal{K}_{\mathbf{X}}\right)}{\lambda_{\min}\left(\Theta_{\mathbf{X}}\right)} \\ &\geq \frac{2\cdot\lambda_{\min}\left(\Theta_{\mathbf{X}}\right)\cdot a_-}{B} - \frac{a_+B}{\lambda_{\min}\left(\Theta_{\mathbf{X}}\right)}.\end{aligned}$$

□

This provides another method of showing the agreement when using σ^2_{NTKGP} for our criterion compared to using σ^2_{NN}.

3.4.2 Incremental Computation of σ^2_{NTKGP}

Computing $\sigma^2_{\text{NTKGP}}\left(\cdot|\mathcal{D}\right)$ for the active learning criterion is computationally expensive. Fortunately, we know that in our active learning algorithm, the labeled set $\mathbf{X}$ is always incremented by one each time, and so, for computing the criterion $\alpha\left(\mathbf{X}\cup\{\mathbf{x}'\}\right)$, we can simply update $\sigma^2_{\text{NTKGP}}\left(\cdot|\mathcal{D}\right)$.

Suppose we let $\mathbf{X}' = \mathbf{X}\cup\{\mathbf{x}'\}$. We can utilize the formula for inversion of block matrices,

$$\begin{bmatrix}\mathbf{A} & \mathbf{B}\\ \mathbf{C} & \mathbf{D}\end{bmatrix}^{-1} = \begin{bmatrix}\left(\mathbf{A}-\mathbf{B}\mathbf{D}^{-1}\mathbf{C}\right)^{-1} & -\mathbf{A}^{-1}\mathbf{B}\left(\mathbf{D}-\mathbf{C}\mathbf{A}^{-1}\mathbf{B}\right)^{-1}\\ -\mathbf{D}^{-1}\mathbf{C}\left(\mathbf{A}-\mathbf{B}\mathbf{D}^{-1}\mathbf{C}\right)^{-1} & \left(\mathbf{D}-\mathbf{C}\mathbf{A}^{-1}\mathbf{B}\right)^{-1}\end{bmatrix},$$

and the matrix inversion lemma [54],

$$\left(\mathbf{A}+\mathbf{B}\mathbf{C}\mathbf{D}\right)^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B}\left(\mathbf{C}^{-1}+\mathbf{D}\mathbf{A}^{-1}\mathbf{B}\right)^{-1}\mathbf{D}\mathbf{A}^{-1} \tag{3.15}$$

to see that

$$\begin{aligned}\Theta_{\mathbf{X}'}^{-1} &= \begin{bmatrix}\Theta_{\mathbf{X}} & \Theta_{\mathbf{X}\mathbf{x}'}\\ \Theta_{\mathbf{x}'\mathbf{X}} & \Theta_{\mathbf{x}'}\end{bmatrix}^{-1}\\ &= \begin{bmatrix}\left(\Theta_{\mathbf{X}} - \Theta_{\mathbf{X}\mathbf{x}'}\Theta_{\mathbf{x}'}^{-1}\Theta_{\mathbf{x}'\mathbf{X}}\right)^{-1} & -\Theta_{\mathbf{X}}^{-1}\Theta_{\mathbf{X}\mathbf{x}'}\left(\Theta_{\mathbf{x}'} - \Theta_{\mathbf{x}'\mathbf{X}}\Theta_{\mathbf{X}}^{-1}\Theta_{\mathbf{X}\mathbf{x}'}\right)^{-1}\\ -\Theta_{\mathbf{x}'}^{-1}\Theta_{\mathbf{x}'\mathbf{X}}\left(\Theta_{\mathbf{X}} - \Theta_{\mathbf{X}\mathbf{x}'}\Theta_{\mathbf{x}'}^{-1}\Theta_{\mathbf{x}'\mathbf{X}}\right)^{-1} & \left(\Theta_{\mathbf{x}'} - \Theta_{\mathbf{x}'\mathbf{X}}\Theta_{\mathbf{X}}^{-1}\Theta_{\mathbf{X}\mathbf{x}'}\right)^{-1}\end{bmatrix}\end{aligned} \tag{3.16}$$

and

$$\begin{aligned}\sigma^2_{\text{NTKGP}}\left(\mathbf{x}|\mathcal{D}\right) &= \Theta_x - \Theta_{\mathbf{x}\mathbf{X}'}\Theta_{\mathbf{X}'}^{-1}\Theta_{\mathbf{X}'\mathbf{x}}\\ &= \sigma^2_{\text{NTKGP}}\left(\mathbf{x}|\mathcal{D}\right) - \Theta_{\mathbf{x}\mathbf{X}}\Theta_{\mathbf{X}}^{-1}\Theta_{\mathbf{X}\mathbf{x}'}\left(\Theta_{\mathbf{x}'} - \Theta_{\mathbf{x}'\mathbf{X}}\Theta_{\mathbf{X}}^{-1}\Theta_{\mathbf{X}\mathbf{x}'}\right)^{-1}\Theta_{\mathbf{x}'\mathbf{X}}\Theta_{\mathbf{X}}^{-1}\Theta_{\mathbf{X}\mathbf{x}}\\ &\quad + 2\cdot\Theta_{\mathbf{x}\mathbf{X}}\Theta_{\mathbf{X}}^{-1}\Theta_{\mathbf{X}\mathbf{x}'}\left(\Theta_{\mathbf{x}'} - \Theta_{\mathbf{x}'\mathbf{X}}\Theta_{\mathbf{X}}^{-1}\Theta_{\mathbf{X}\mathbf{x}'}\right)^{-1}\Theta_{\mathbf{x}'\mathbf{x}}\\ &\quad - \Theta_{\mathbf{x}\mathbf{x}'}\left(\Theta_{\mathbf{x}'} - \Theta_{\mathbf{x}'\mathbf{X}}\Theta_{\mathbf{X}}^{-1}\Theta_{\mathbf{X}\mathbf{x}'}\right)^{-1}\Theta_{\mathbf{x}'\mathbf{x}}.\end{aligned}$$

Given that $\Theta_{\mathbf{X}}^{-1}$ can also be reused from the previous round and updated iteratively using Equation 3.16, this means the variance can be computed more efficiently.

3.4.3 Sparse GP Approximation of Σ_{NTKGP}

Another benefit of using Σ_{NTKGP} for approximating the EV-GP criterion is that, unlike Σ_{NN}, the covariance matrices in the form of Σ_{NTKGP} are well-studied in GP literature (Section 2.3), and methods of sparsifying the kernel matrix have been proposed (Subsection 2.3.2). Below, we discuss how these methods can be used.

Given a set of labeled training data $\mathcal{D} = (\mathbf{X}, \mathbf{y})$, we want to compute the posterior variance $\Sigma_{\text{NTKGP}}\left(\mathbf{X}_T|\mathcal{D}\right)$. Let $\mathbf{U}$ be a set of inducing points which is representative of $\mathbf{X}$. In general, $\mathbf{U}$ does not have to be a subset of $\mathbf{X}$, and therefore is not necessarily a valid candidate for the active set. For simplicity, we will apply the Fully-Independent Conditional (FIC) approximation, where we assume that $\mathbf{y}$ and $\mathbf{y}_T$ are conditionally independent given $\mathbf{u} \coloneqq f(\mathbf{U})$:

$$p\left(\mathbf{y}_T|\mathbf{y}\right) = \int p\left(\mathbf{y}_T|\mathbf{u},\mathbf{y}\right)p\left(\mathbf{u}|\mathbf{y}\right)d\mathbf{u} \approx \int p\left(\mathbf{y}_T|\mathbf{u}\right)p\left(\mathbf{u}|\mathbf{y}\right)d\mathbf{u}$$

We assume that $p\left(\mathbf{u}|\mathbf{y}\right) \approx q\left(\mathbf{u}\right)$ and the goal is to compute the distribution $q\left(\mathbf{u}\right)$ such that $D_{KL}[q\left(\mathbf{u}\right)\|p\left(\mathbf{u}|\mathbf{y}\right)]$ is minimised.

Assume that $q\left(\mathbf{u}\right)$ is a normal distribution, i.e., $\mathbf{u} \overset{\text{FIC}}{\sim} \mathcal{N}\left(\mu_{\mathbf{U}}, \Sigma_{\mathbf{U}}\right)$ where the mean and the covariance function are dependent of our choice of set $\mathbf{X}$. Then, minimizers, $\mu_{\mathbf{U}}$ and $\Sigma_{\mathbf{U}}$, of the KL divergence also maximise the expected lower bound (ELBO) [24]:

$$L\left(q\right) = \int q\left(\mathbf{u}\right)\left[\int q\left(\mathbf{f}|\mathbf{u}\right)\log\frac{p\left(\mathbf{f},\mathbf{y}|\mathbf{u}\right)}{q\left(\mathbf{f}|\mathbf{u}\right)}d\mathbf{f}\right]d\mathbf{u} - D_{KL}[q\left(\mathbf{u}\right)\|p\left(\mathbf{u}\right)] \tag{3.17}$$

where $\mathbf{f}$ is the underlying (latent) function, i.e., the noiseless version of $\mathbf{y}$.

In order to compute the distribution $q(\mathbf{u})$ which maximises L, we can compute its gradient. In this case the gradient has a nice closed form:

$$\frac{\partial L}{\partial \mu_{\mathbf{U}}} = \Theta_{\mathbf{U}}^{-1}\mu_{\mathbf{U}} - \sum_{(\mathbf{x},y)\in\mathcal{D}} \left[G(\mathbf{U},\mathbf{x}) - F(\mathbf{U},\mathbf{x})\,\mu_{\mathbf{U}}\right] \tag{3.18}$$

$$\frac{\partial L}{\partial \Sigma_{\mathbf{U}}} = \Sigma_{\mathbf{U}}^{-1} - \frac{1}{2}\Theta_{\mathbf{U}}^{-1} - \frac{1}{2}\sum_{\mathbf{x}\in\mathbf{X}} F(\mathbf{U},\mathbf{x}) \tag{3.19}$$

where

$$\begin{aligned} G(\mathbf{U},\mathbf{x}) &= \Theta_{\mathbf{U}}^{-1}\Theta_{\mathbf{U}\mathbf{x}}\Gamma_{\mathbf{x}}^{-1}y \\ F(\mathbf{U},\mathbf{x}) &= \Theta_{\mathbf{U}}^{-1}\Theta_{\mathbf{U}\mathbf{x}}\Gamma_{\mathbf{x}}^{-1}\Theta_{\mathbf{x}\mathbf{U}}\Theta_{\mathbf{U}}^{-1}\mu_{\mathbf{U}} \\ \Gamma_{\mathbf{x}} &= \Theta_{\mathbf{x}} - \Theta_{\mathbf{x}\mathbf{U}}\Theta_{\mathbf{U}}^{-1}\Theta_{\mathbf{U}\mathbf{x}} \end{aligned}$$

We will also let $\Theta_{\mathbf{U}} = \Theta(\mathbf{X}_{\mathbf{U}},\mathbf{X}_{\mathbf{U}})$ and $\Theta_{\mathbf{x}\mathbf{U}} = \Theta(\mathbf{x},\mathbf{X}_{\mathbf{U}})$. Using the closed form expression of the gradient, the optimal $\mu_{\mathbf{U}}$ and $\Sigma_{\mathbf{U}}$ can be computed through gradient ascent or by directly solving equations $\partial L/\partial\mu_{\mathbf{U}} = 0$ and $\partial L/\partial\Sigma_{\mathbf{U}} = 0$.

Given the optimal $\mu_{\mathbf{U}}^*$ and $\Sigma_{\mathbf{U}}^*$, the sparse NTKGP approximation is given as

$$\begin{aligned} \mu_{\text{sNTKGP}}(\mathbf{y}_T|\mathbf{y}) &= \Theta_{T\mathbf{U}}\Theta_{\mathbf{U}}^{-1}\mu_{\mathbf{U}}^* \\ \Sigma_{\text{sNTKGP}}(\mathbf{y}_T|\mathbf{y}) &= \Theta_T - \Theta_{T\mathbf{U}}\Theta_{\mathbf{U}}^{-1}\Theta_{\mathbf{U}T} + \Theta_{T\mathbf{U}}\Theta_{\mathbf{U}}^{-1}\Sigma_{\mathbf{U}}^*\Theta_{\mathbf{U}}^{-1}\Theta_{\mathbf{U}T} \end{aligned}$$

The approximate posterior can now be computed in $\mathcal{O}(m^3)$ time, where m is the number of inducing points, instead of $\mathcal{O}(n^3)$. The equation can also be iteratively computed in quadratic time if Equation 3.15 is used for matrix inversion.

From the decomposition in Equation 3.18 and Equation 3.19, we see that the FIC approximation is nice to use since it is possible to decompose the summation above based on terms $F(\mathbf{U},\mathbf{x})$ and $G(\mathbf{U},\mathbf{x})$ where both terms depend only on one element, independent of the other elements in $\mathbf{X}$. This is convenient when trying to add one element and see what effect it has on the inducing points distribution. Using this fact, it is also possible to apply a further linear approximation on the criterion which allows for even more efficient computation.

3.4.4 Approximating Incremental Change in Σ_{NTKGP}

Given the inducing point prior $\Sigma_{\mathbf{U}}(\mathbf{X})$ as computed in Subsection 3.4.3, we can theoretically compute the approximate posterior $\Sigma_{\text{sNTKGP}}(\mathbf{X}_T|\mathbf{X}\cup\{\mathbf{x}'\})$ when one point is added directly using the same method. We propose a simple technique to incrementally update the value of $\Sigma_{\mathbf{U}}(\mathbf{X}\cup\{\mathbf{x}'\})$, and therefore $\Sigma_{\text{sNTKGP}}(\mathbf{X}_T|\mathbf{X}\cup\{\mathbf{x}'\})$, when a small number of points are added to the training set. The approximation technique is akin to the use of influence functions [31].

When we add a single point to the training set, the ELBO (Equation 3.17) will only change by a small amount. Let $L' = L + \Delta L$ be the new loss function after adding $\mathbf{x}'$ into the set $\mathbf{X}$, and ΔL be the contribution specifically from the new element $\mathbf{x}'$. Since L only changes by an amount ΔL, we do not expect $\Sigma_{\mathbf{U}}$ to change by much. Let $\Sigma_{\mathbf{U}}(\mathbf{X} \cup \{\mathbf{x}'\}) = \Sigma_{\mathbf{U}}(\mathbf{X}) + \Delta\Sigma_{\mathbf{U}}(\mathbf{x}'; \mathbf{X})$ where $\Delta\Sigma_{\mathbf{U}}(\mathbf{x}'; \mathbf{X})$ is the change of the inducing prior after adding the training point $\mathbf{x}'$. We see that $\Sigma_{\mathbf{U}}(\mathbf{X} \cup \{\mathbf{x}'\})$ and $\Sigma_{\mathbf{U}}(\mathbf{X})$ would be slightly different from each other.

We find that the change in the inducing points posterior can be instead given by

$$\Delta\Sigma_{\mathbf{U}}(\mathbf{x}'; \mathbf{X}) \approx -\left(\frac{\partial^2 L}{\partial \Sigma^2}|_{\Sigma=\Sigma_{\mathbf{U}}(\mathbf{X})}\right)^{-1}\left(\frac{\partial(\Delta L)}{\partial \Sigma}|_{\Sigma=\Sigma_{\mathbf{U}}(\mathbf{X})}\right) \tag{3.20}$$

Conveniently, the FIC approximation gives rise to a loss function whose gradient is easily decomposable. We can write the derivative of the loss function contribution from $\mathbf{x}'$ as

$$\frac{\partial(\Delta L)}{\partial \Sigma} = -\frac{1}{2}F(\mathbf{U}, x) \tag{3.21}$$

We also see that the Hessian of the original loss is equal to the Jacobian of the matrix inverse, i.e.

$$\frac{\partial^2 L}{\partial \Sigma^2} = \frac{\partial}{\partial \Sigma}\left(\frac{1}{2}\Sigma^{-1}\right) \tag{3.22}$$

which can easily be computed in closed form and also using any auto-differentiation package. Expressions from Equation 3.21 and Equation 3.22 can be plugged back into Equation 3.20 to obtain the change of $\Sigma_{\mathbf{U}}$ when one new sample is added.

Therefore, any criterion α we have which would depend on the approximation of the sNTKGP can be approximated as

$$\Delta\alpha(\mathbf{x}'; \mathbf{X}) \approx \left\langle \frac{\partial \alpha}{\partial \Sigma_{\mathbf{U}}}, \Delta\Sigma_{\mathbf{U}}(\mathbf{x}'; \mathbf{X}) \right\rangle$$

This expression can be used to compute the change in the active learning criterion when an additional data point is added. Because this term is based on the inner product of some matrix quantities, it is parallelizable and in practice speeds up the algorithm by a large amount.

Experimentally, the Hessian can be expensive to compute. In order to speed up the computation, we can instead use an alternate parametrization of the sparse GP called the *natural parameters* which defines parameters to match more naturally to the terms that appears in the Gaussian distribution. We refer the readers to [24] for further details on this.

3.4.5 Other Criterions

Variance Percentile

Another possible way to use σ^2_{NTKGP} for creating an AL criterion is use the rth percentile of variance in the test points, i.e.

$$\alpha_{\text{V}}\left(\mathbf{X}; r\right) = -\text{percentile}\left(\{\sigma^2_{\text{NTKGP}}\left(\mathbf{x}|\mathcal{D}\right) : \mathbf{x} \in \mathbf{X}_T\}, r\right).$$

In this case, letting $r = 50$ is equivalent to considering the median of test output variance, and letting $r = 100$ is equivalent to considering the maximum test output variance. Unfortunately, the percentile function is not submodular in general, and therefore has no theoretical guarantees when points are selected using the greedy algorithm.

Mutual Information Criterion

The criteria discussed thus far often do not take into account the distribution of $\mathbf{X}_T$. Through the lens of information theory, we can view the active learning problem as attempting to select points $\mathbf{X}$ with outputs $\mathbf{y}$ such that the most information can be obtained about $\mathbf{y}_T$. As in [33], we can attempt to maximise the mutual information

$$\begin{aligned}
\alpha_{\text{MI}}\left(\mathbf{X}_L\right) &= \mathbb{I}\left[\mathbf{y}_L; \mathbf{y}_{T\setminus L}\right] \\
&= \mathbb{H}\left[\mathbf{y}_{T\setminus L}\right] - \mathbb{H}\left[\mathbf{y}_{T\setminus L}|\mathbf{y}_L\right] \\
&= \frac{1}{2}\log\det\left(\Sigma_{\text{NTKGP}}\left(\mathbf{X}_{T\setminus L}\right)\right) - \frac{1}{2}\log\det\left(\Sigma_{\text{NTKGP}}\left(\mathbf{X}_{T\setminus L}|\mathcal{D}\right)\right) + \text{constant},
\end{aligned}$$

where we use the shorthand $\mathbf{X}_{T\setminus L} = \mathbf{X}_T \setminus \mathbf{X}_L$. We are able to arrive at the final line since we know the predictive covariance follows a multivariate Gaussian distribution.

In the case that the test set and the unlabeled pool are disjoint, we have $\mathbf{y}_{T\setminus L} = \mathbf{y}_T$, which means $\mathbb{H}\left[\mathbf{y}_{T\setminus L}\right]$ is a constant regardless of which active set we choose. In this case, maximising α_{MI} is equivalent to minimising $\mathbb{H}\left[\mathbf{y}_T|\mathbf{y}_L\right]$.

A particular issue with using the mutual information is the numerical stability when computing the entropy values. In particular, α_{MI} involves computing the entropy $\mathbb{H}\left[\mathbf{y}_{T\setminus L}|\mathbf{y}_L\right]$, which involves computing the determinant $\det\Sigma_{\text{NN}}\left(\mathbf{X}_{T\setminus L}|\mathcal{D}\right)$. However, when the test set $\mathbf{X}_{T\setminus L}$ have points which are highly correlated with each other, the matrix $\Sigma_{\text{NN}}\left(\mathbf{X}_{T\setminus L}|\mathcal{D}\right)$ may be singular and cause the determinant to be undefined.

Notice that the decision of using $\mathbf{X}_{T\setminus L}$ instead of $\mathbf{X}_T$ will already alleviate some of the issues regarding singular matrices. However, in order to prevent further issues, in our experiments, we decide to pre-filter the points that are used for the test set. In particular, instead of using all points from the test set, we select only a subset of training points such that $\mathbb{H}\left[\mathbf{y}_T\right]$ is maximised. This means that the subset selected will be as independent from each other as possible. Furthermore, using a subset of points instead of the full test set also reduces the matrix size, which speeds up our computation as well. The subset of points that

are selected are done so using the K-Means++ initialization method. Furthermore, to avoid computing the determinant of a singular matrix, we add in a diagonal noise term in order to compute the determinant of $\Sigma_{\text{NN}}\left(\mathbf{X}_{T\setminus L}|\mathcal{D}\right)+\sigma^2 I$ instead. This corresponds to the case where the observation has some added noise with variance σ^2.

3.5 AL Algorithm

Algorithm 1 EV-GP

Input: Initial labeled data $(\mathbf{X}_0, \mathbf{y}_0)$, unlabeled pool $\mathbf{X}$, batch size b
$(\mathbf{X}_L, \mathbf{y}_L) \leftarrow (\mathbf{X}_0, \mathbf{y}_0)$
repeat
  for b iterations **do**
    $x^* \leftarrow \arg\max_{x\in\mathbf{X}\setminus\mathbf{X}_L} \alpha_{\text{EV}}\left(\mathbf{X}_L \cup \{x\}\right)$
    $\mathbf{X}_L \leftarrow \mathbf{X}_L \cup \{x^*\}$
  end for
  Query the unlabeled points in $\mathbf{X}_L$ for the labels $\mathbf{y}_L$
until budget exhausted
return $(\mathbf{X}_L, \mathbf{y}_L)$

We propose a greedy approach, which is guaranteed to give a $\left(1-\frac{1}{e}\right)$-optimal solution [46], and leave the use of other more sophisticated submodular optimization techniques to future works. The greedy algorithm requires $\mathcal{O}(nk)$ criterion computations, where n is the size of the unlabeled pool and k is the AL budget, which can be slow. There is a large literature on efficient submodular maximization algorithms with cardinality constraints. This, however, is not the focus of our work. Nonetheless, for practical purposes, we propose two simple optimization techniques.

1. The first optimization technique is based on the Accelerated-Greedy algorithm [40]. For each element in the unlabeled data pool, we store the marginal gain value,

$$\Delta_{\mathbf{x}}(\mathbf{X}) \coloneqq \alpha_{\text{EV}}\left(\mathbf{X}\cup\{\mathbf{x}\}\right) - \alpha_{\text{EV}}\left(\mathbf{X}\right),$$

   for some $\mathbf{X}$ that was previously active. Then, as the greedy algorithm proceeds, we will continue to grow the active set into some $\mathbf{X}' \supset \mathbf{X}$. From submodularity, we know that $\Delta_{\mathbf{x}}(\mathbf{X}) \geq \Delta_{\mathbf{x}}(\mathbf{X}')$. This means that in a particular round with active set $\mathbf{X}'$, if we already have computed the marginal gain, $\Delta_{\mathbf{x}'}(\mathbf{X}')$, for some $\mathbf{x}'$, and $\Delta_{\mathbf{x}'}(\mathbf{X}') \geq \Delta_{\mathbf{x}}(\mathbf{X})$, then there is no point in computing with $\mathbf{x}$ since

$$\begin{aligned}&\Delta_{\mathbf{x}'}(\mathbf{X}') \geq \Delta_{\mathbf{x}}(\mathbf{X}) \geq \Delta_{\mathbf{x}}(\mathbf{X}')\\ \Rightarrow\ &\alpha_{\text{EV}}\left(\mathbf{X}'\cup\{\mathbf{x}'\}\right) \geq \alpha_{\text{EV}}\left(\mathbf{X}'\cup\{\mathbf{x}\}\right).\end{aligned}$$

   However, when we use the empirical NTK in our experiments, we reinitialize the neural

network after each batch of AL which changes the empirical NTK, and hence, we need to recompute the criterion values.

2. Another optimization technique is the Stochastic-Greedy algorithm [41], where we calculate the criterion for a random subset of points, $\mathbf{X}_R \subset \mathbf{X}_U$ in each round. This technique is able to achieve accuracy arbitrarily close to $1 - 1/e$ in $\mathcal{O}(n)$ time (independent of k).

Chapter 4

Experiments

Although the theoretical properties of Σ_{NN} and Σ_{NTKGP} are applicable to infinite-width neural networks, we follow the practice of previous works on NTKs [21, 42] and use finite-width neural networks, since are able to achieve good performances. In our experiments, we test our algorithm using both the theoretical NTKs, computed using the Jax-based [6] Neural-Tangents package [48], and the empirical NTK computed using PyTorch. We will use EV-GP-Emp to denote instances when we use the empirical NTK for our algorithm.

We compare our algorithm with previous baselines which require minimal model training between different batches and do not incur significant extra computations – random selection, K-Means++ [3], BADGE [4] and MLMOC [42] algorithms. The former two, like EV-GP, can select all the points in a single batch, but the latter two are not designed for such a setting, and need modification for this purpose.

When reporting the output variance or performance metrics, we train a neural network 50 times for regression tasks and 25 times for classification tasks with the same architecture but with different model initializations. All experiments are repeated 5 times (unless stated otherwise) and the mean and standard deviations of the above-mentioned scores are reported. We adopt the MSE loss Equation 2.9 for regression experiments and the cross-entropy loss for classification experiments, which is consistent with previous works on NTK [60].

Some experimental details, as well as descriptions of the datasets and the algorithms, have been deferred to the appendix.

4.1 Points Chosen by EV-GP

In this section, we explore the attributes of the active set selected using the EV-GP criterion. From Figure 4.1, we see that the output variance with respect to model initialization is heteroscedastic. This means that some inputs have more varied model prediction than others. Unlike other coreset methods, Algorithm 1 does not necessarily pick points which evenly cover the whole input space, but instead prioritizes points from regions with high predictive variance, and balances the active set with points from dense regions.

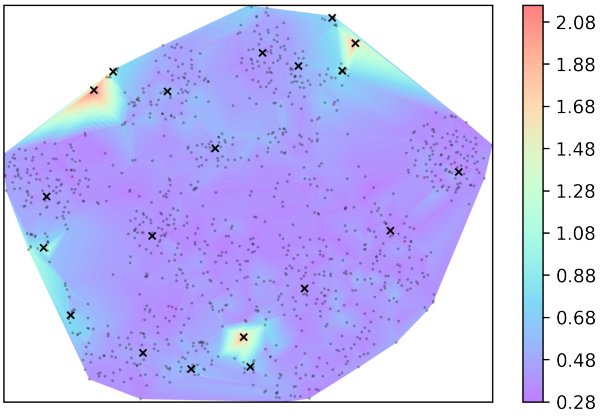

Figure 4.1: Visualization of points selected by Algorithm 1. The black points represent images in the Handwritten Digits dataset projected onto 2 dimensions using t-SNE. The contours represent the predictive standard deviation computed empirically using the model output at initialization. The crosses represent the points selected by the algorithm.

4.2 Correlation Between σ^2_{NTKGP} and Output Variance

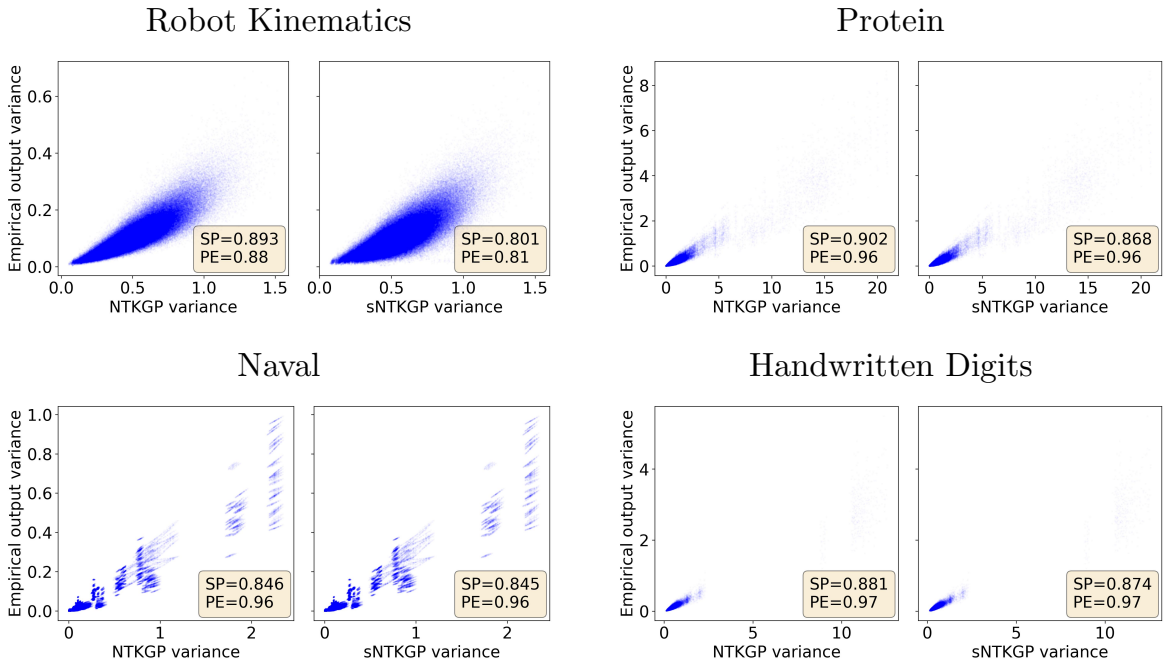

Figure 4.2: The empirical variance (with respect to random network initialization) against the approximate prediction variance $\sigma^2_{\text{NTKGP/sNTKGP}}$. SP refers to the Spearman rank correlation coefficient and PE refers to the Pearson correlation coefficient.

Here we study whether our approximate output variance σ^2_{NTKGP} (Section 3.1) can accurately capture the output variance of neural networks (w.r.t. the random initializations),

and hence, the initialization robustness. Figure 4.2 verifies that σ^2_{NTKGP} is highly correlated with the observed output variance of the neural network and the variances are generally confined within some region, which provides an empirical corroboration for Theorem 3.2. This justifies our choice of using σ^2_{NTKGP} to measure the output variance w.r.t. model initialization, and hence, initialization robustness. Additionally, we see that σ^2_{sNTKGP}, which is more computationally efficient (Subsection 3.4.3), is also highly correlated with the observed output variance, albeit lower than its full-rank counterpart.

4.3 Experiments on Regression Tasks

Here, we evaluate our EV-GP criterion (Section 3.4) on regression tasks. In the experiments here, each algorithm is given an unlabeled pool of data and no initial labeled data, and all methods use a 2-layer MLP with ReLU activation. When reporting the EV-GP criterion values, we will ignore the $\sigma^2_{\text{NN}}(\mathbf{x}|\phi)$ terms and instead report the average $-\sigma^2_{\text{NN}}(\mathbf{x}|\mathcal{D})$.

4.3.1 Sequential Data Selection

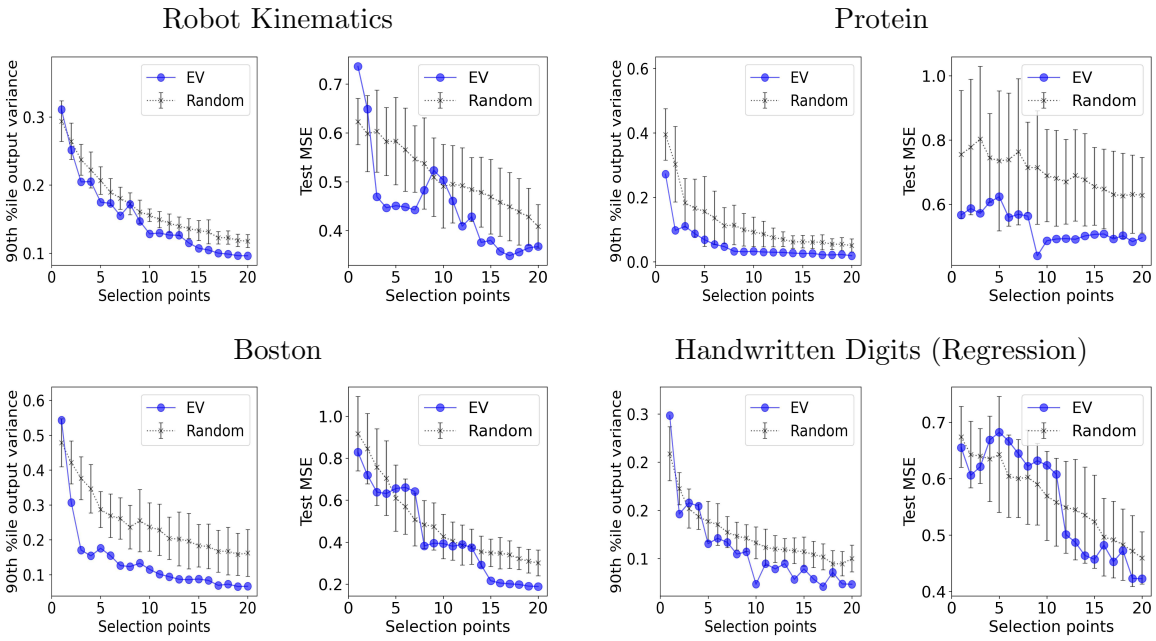

Figure 4.3: The 90$^{\text{th}}$ percentile of prediction variance and the test loss against the size of the active set in a sequential data selection setting.

In Figure 4.3, our EV-GP criterion is used to sequentially select the data points (i.e. the batch size, k, is 1). The left plot for each dataset shows that sequentially maximizing our EV-GP criterion indeed leads to the selection points which progressively reduce the output variance, and our AL algorithm consistently outperforms a random selection strategy. The right plot shows that the points selected by maximizing the EV-GP criterion also sequentially reduce the test MSE and hence improve the predictive performance of the neural network.

Figure 4.3 provides an empirical justification for Theorem 3.4, which theoretically showed that minimizing the approximate output variance σ_{NTKGP} (which is achieved by maximizing our EV-GP criterion) also improves the generalization performance of overparameterized neural networks.

4.3.2 Batched Data Selection

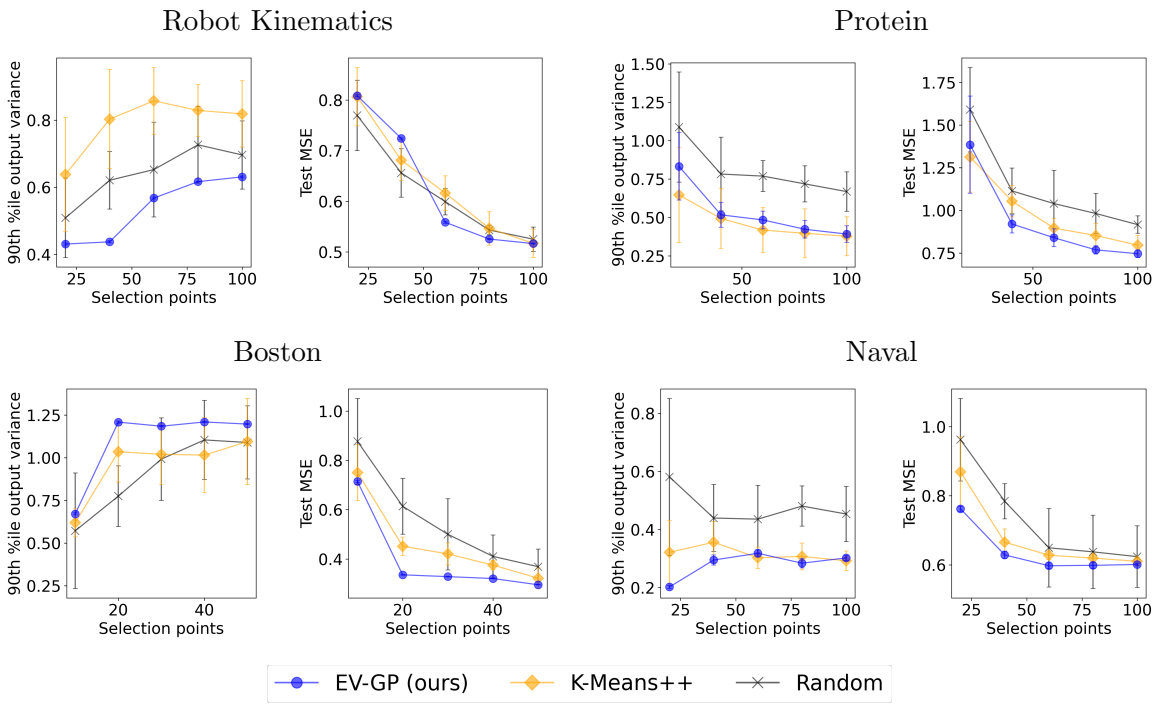

Figure 4.4: The 90$^{\text{th}}$ percentile of prediction variance and the test loss against the size of the active set in a batch data selection setting.[2]

We also tested our EV-GP criterion in the more practical AL setting where a batch of points (here, 20) are selected in every round. Figure 4.4 shows that in the batch setting, our EV-GP criterion is still able select batches of points which lead to both low output variance (left column) and small test error (right column), and outperforms the other baselines.

4.3.3 Evidence for Theorem 3.4

Figure 4.5 shows that an easier regression task leads to a larger degree of correlation between the output variance and the test error. This is consistent with Theorem 3.4 which suggests that an easier task (or, a simpler groundtruth function), indicated by a smaller B, implies a low value of ζ, increasing the degree of correlation between the output variance and the generalization error. We note that even in tasks exhibiting low correlation between EV-GP and MSE loss, Algorithm 1 is effective (Figure 4.4).

[2]We omit BADGE and MLMOC algorithms from this experiment since they are not designed for regression tasks.

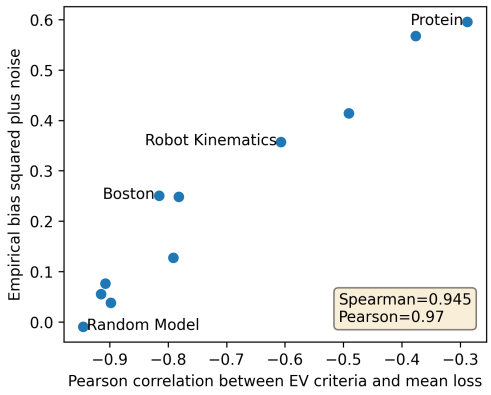

Figure 4.5: The empirical bias (defined as the average MSE loss minus the output variance) against the correlation of EV-GP with the MSE loss. Note that EV-GP is higher when the output variance and the MSE loss are lower, which is why the correlation values on the x-axis are negative.

4.3.4 Sparse Approximations

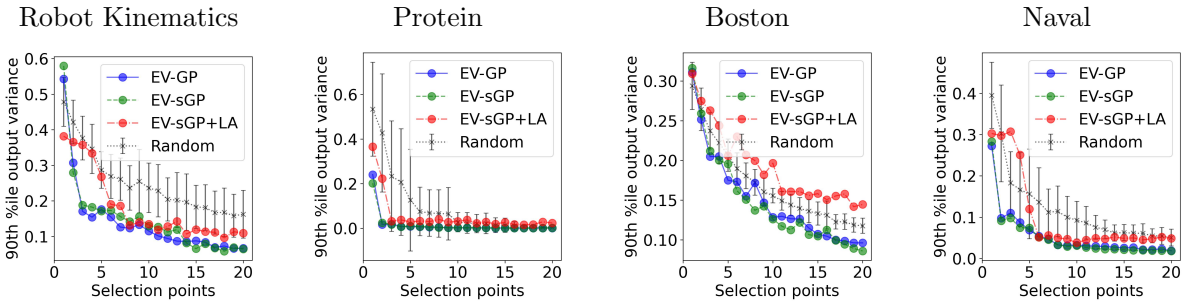

Figure 4.6: The 90$^{\text{th}}$ percentile of prediction variance in a sequential data selection setting using different approximations of σ_{NTKGP}.

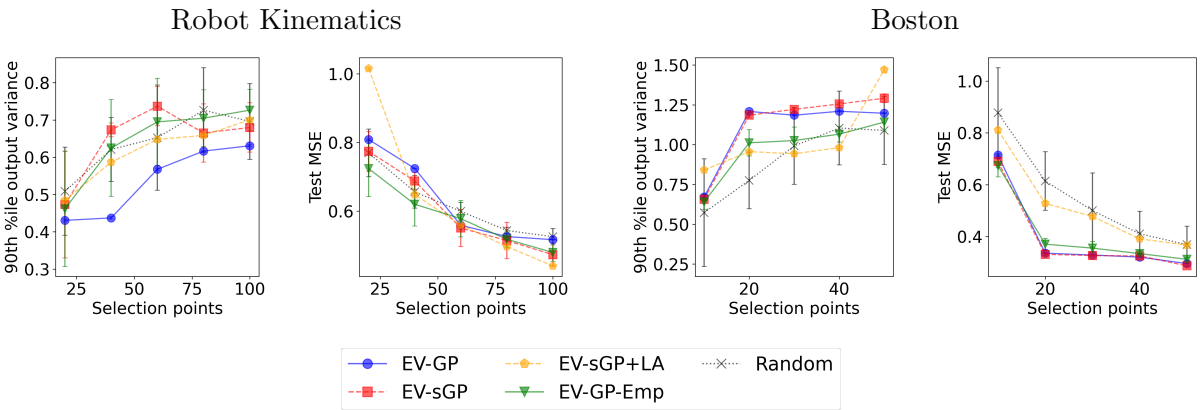

Figure 4.7: The 90$^{\text{th}}$ percentile of prediction variance and mean test loss in a batched data selection setting using different approximations of σ_{NTKGP}.

In Figure 4.6 and Figure 4.7, we present the results for various approximation methods of the NTKGP – approximating the NTK empirically, and approximating the covariance using sparse GP techniques (as introduced in Subsection 3.4.3). We can see that the empirical approximation of the NTK sees a drop in performance compared to the theoretical NTK. However, it still provides a useful enough approximation for our purposes. Moreover, we can see that the sNTKGP approximation provides a useful approximation for the criterion as well. Applying a further linear approximation, however, to the sNTKGP computation worsens the selection process. This suggests that the covariance function from the sNTKGP may not be smooth enough for a linear approximation method to be precise.

4.3.5 Other Criteria

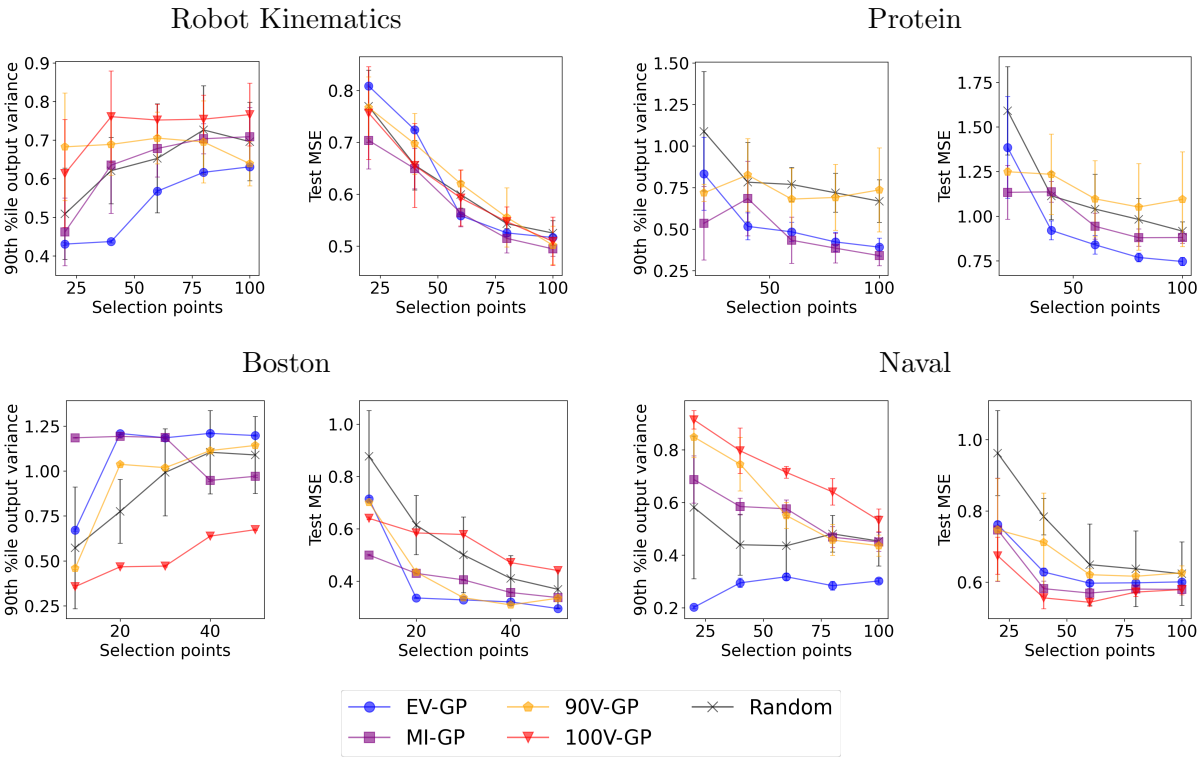

Figure 4.8: Active learning on regression datasets with different criteria based on σ^2_{NTKGP} (Subsection 3.4.5).

Here, we show the results for experiments with other active learning criteria, as presented in Subsection 3.4.5. In Figure 4.8 we show the results in the batch setting. We find that $\alpha_{100\text{V}}$ tends to perform poorly compared to the other criteria, and even worse than Random in some instances. Since $\alpha_{100\text{V}}$ tends puts too much focus on reducing the variance of outliers, it does not provide a diverse subset of points. Reducing this threshold to the 90^{th} percentile (90V-GP) is able to give a better subset, however, still not as effective as EV-GP. Meanwhile, MI-GP often has good performance, sometimes even outperforming EV-GP. However, we find it is still less preferable since its advantage over EV-GP is not consistent and also due to the aforementioned issue of computational efficiency (Subsection 3.4.5).

4.4 Experiments on Classification Tasks

Here we use our EV-GP criterion for classification tasks. We use a wider variety of neural network architectures, including multi-layer perceptron (MLP) networks with ReLU activation, convolutional neural networks (CNNs) and WideResNet [72], as in [42].

4.4.1 Performance Comparison

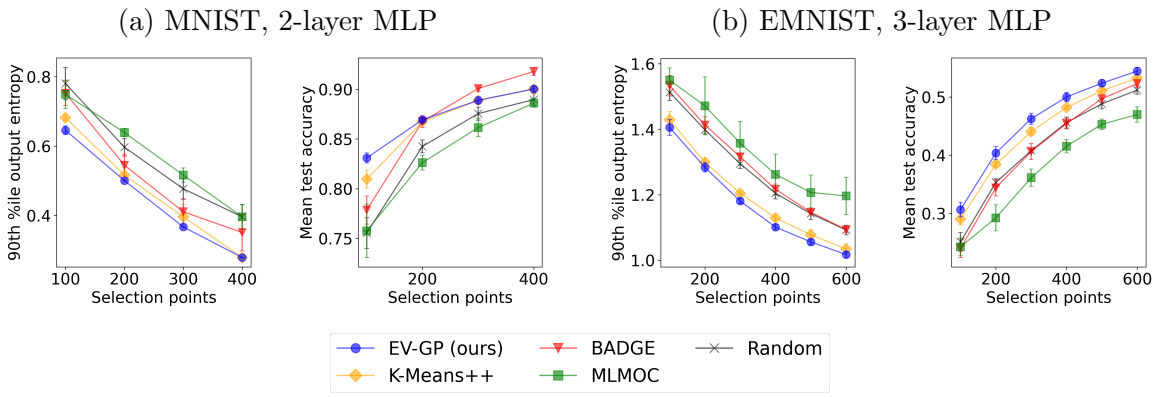

Figure 4.9: Output entropy and mean test accuracy in classification experiments using MLP networks.

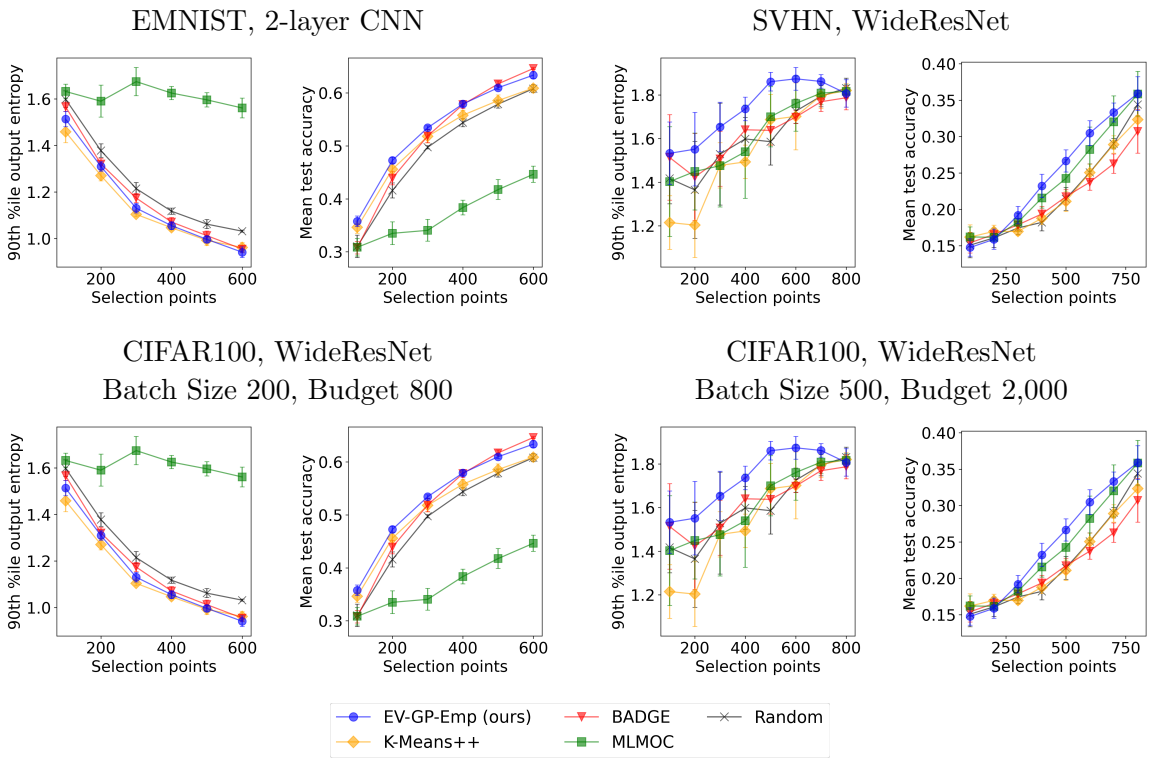

Figure 4.10: Output entropy and mean test accuracy in classification experiments using CNNs.

Figure 4.9 presents the comparison of our EV-GP criterion with other baselines, in which all methods use MLPs. The figures show that our EV-GP criterion is indeed able to select points which lead to both initialization robustness (i.e., low output entropy plotted in the first column) and good generalization performances (i.e., high test accuracy shown in the second column). The results are consistent with those for the regression tasks (Section 4.3). Moreover, our EV-GP criterion outperforms the other baselines, especially in the earlier rounds when there is a small number of selected points. Figure 4.10 plots the results using more sophisticated neural network architectures (i.e., CNNs and WideResNets), in which our EV-GP criterion also consistently outperforms the other baselines in terms of the test accuracy.

4.4.2 Effect of Batch Size

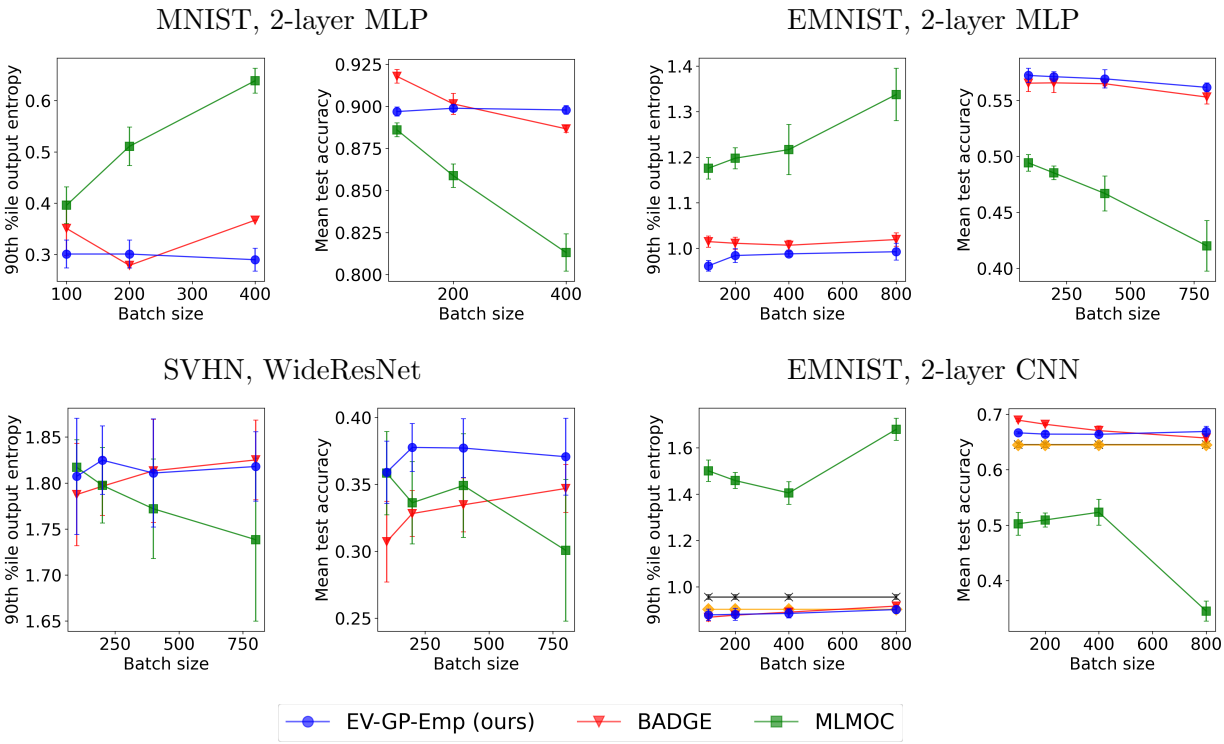

Figure 4.11: Results on classification tasks with varying batch sizes.

Here we examine the impact of the batch size on the performances of different AL algorithms, by fixing the total query budget and varying the batch size. The results in Figure 4.11 show that EV-GP-Emp (which uses the empirical NTK) is minimally affected by increasing batch size[3]. This is reasonable since EV-GP-Emp does not require labels. Therefore, batch size has no effect on the performance. In contrast, when the batch size is increased, MLMOC experiences a large drop in performance for both datasets and the performance of BADGE is significantly decreased for MNIST. This may be mainly attributed to their reliance on the

[3]We omit Random, K-Means++ and EV-GP from the graph since the selection algorithm is independent of batch size.

labels, because a larger batch size reduces the frequency of the availability of, and hence their abilities to use, labels. Moreover, another factor which causes the detrimental effect of a larger batch size on MLMOC is that a larger batch size is likely to reduce the diversity of the selected points [42].

4.4.3 Effect of Network Width

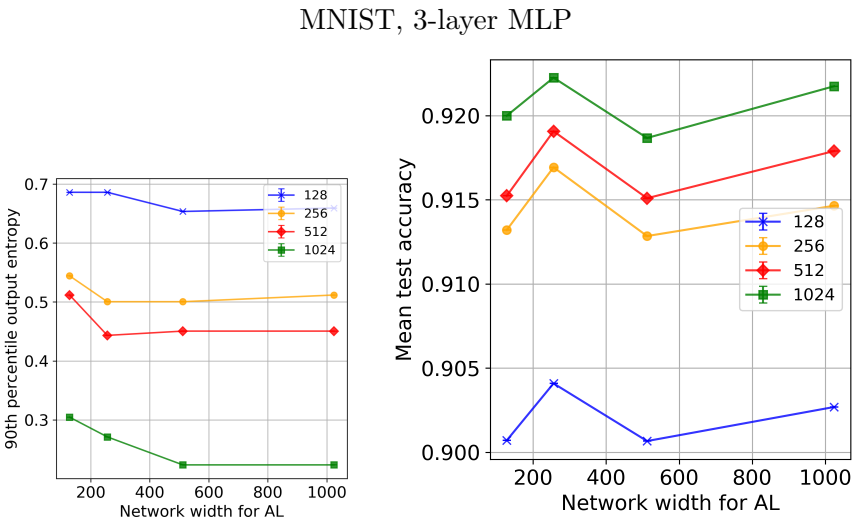

Figure 4.12: Performances of our EV-GP-Emp with varying network widths for data selection (x-axis) and network training (different plots).

Here we investigate how the width of the network affects the performance of our EV-GP-Emp algorithm, by changing the widths of both, the network used for data selection (i.e., for calculating our EV-GP criterion) and the one used for training. The left column in Figure 4.12 shows that increasing the width of either of the networks improves initialization robustness. This is because a wider network for both data selection and training can reduce the approximation error in using σ^2_{NTKGP} and therefore, improve the accuracy of approximation for our EV-GP criterion. As a result, this leads to more accurate data selection and hence better initialization robustness. The right column shows that increasing the width of the network for training improves test accuracy, which can be attributed to the better expressivity of wider networks. However, increasing the width of the network used for data selection does not have a significant impact on the test accuracy. This suggests that the network does not need to be extremely wide in order to select a good active set that leads to good predictive performance of the trained network.

Chapter 5

Conclusion

We have introduced a computationally efficient and theoretically grounded criterion for neural active learning, which can lead to the selection of points that result in both initialization robustness and good generalization performances. Extensive empirical results have shown that our criterion is highly correlated to both the initialization robustness and generalization error, and that it consistently outperforms existing baselines. An interesting future direction is to incorporate our algorithm to select the initial points for other neural active learning algorithms to further enhance their performances, because our algorithm has shown impressive performances in scenarios with limited initial labeled data. Moreover, our theoretical analysis was limited to MSE loss. Since our algorithm performs well with cross-entropy loss as well (demonstrated in Section 4.4), future works can try to prove the theorems for it. Finally, our proposed criteria is quite simple, and other AL criteria can also be constructed and studied using the output variance as modeled by the NTK.

Bibliography

[1] Moloud Abdar et al. "A Review of Uncertainty Quantification in Deep Learning: Techniques, Applications and Challenges". In: *Information Fusion* 76 (Dec. 2021), pp. 243–297. ISSN: 15662535. DOI: 10.1016/j.inffus.2021.05.008. arXiv: 2011.06225 [cs].

[2] Sanjeev Arora et al. "On Exact Computation with an Infinitely Wide Neural Net". In: *arXiv:1904.11955 [cs, stat]* (Nov. 2019). arXiv: 1904.11955 [cs, stat].

[3] David Arthur and Sergei Vassilvitskii. "K-Means++: The Advantages of Careful Seeding". In: (), p. 11.

[4] Jordan T. Ash et al. "Deep Batch Active Learning by Diverse, Uncertain Gradient Lower Bounds". In: *arXiv:1906.03671 [cs, stat]* (Feb. 2020). arXiv: 1906.03671 [cs, stat].

[5] Shai Ben-David et al. "A Theory of Learning from Different Domains". In: *Machine Learning* 79.1-2 (May 2010), pp. 151–175. ISSN: 0885-6125, 1573-0565. DOI: 10.1007/s10994-009-5152-4.

[6] James Bradbury et al. *JAX: Composable Transformations of Python+NumPy Programs*. 2018.

[7] John Bridle. "Training Stochastic Model Recognition Algorithms as Networks can Lead to Maximum Mutual Information Estimation of Parameters". In: *Advances in Neural Information Processing Systems*. Ed. by D. Touretzky. Vol. 2. Morgan-Kaufmann, 1989. URL: https://proceedings.neurips.cc/paper_files/paper/1989/file/0336dcbab05b9d5ad24f4333c7658a0e-Paper.pdf.

[8] Davide Cacciarelli and Murat Kulahci. *A survey on online active learning*. 2023. arXiv: 2302.08893 [stat.ML].

[9] Rita Chattopadhyay et al. "Batch Mode Active Sampling Based on Marginal Probability Distribution Matching". In: *KDD : proceedings / International Conference on Knowledge Discovery & Data Mining. International Conference on Knowledge Discovery & Data Mining* 2012 (2012), pp. 741–749. ISSN: 2154-817X. DOI: 10.1145/2339530.2339647.

[10] Sayak Ray Chowdhury and Aditya Gopalan. "On kernelized multi-armed bandits". In: *Proc. ICML*. 2017, pp. 844–853.

[11] Gregory Cohen et al. *EMNIST: an extension of MNIST to handwritten letters*. arXiv:1702.05373 [cs]. Mar. 2017. URL: http://arxiv.org/abs/1702.05373 (visited on 01/26/2023).

[12] Eric C. Cyr et al. *Robust Training and Initialization of Deep Neural Networks: An Adaptive Basis Viewpoint*. Dec. 2019. arXiv: 1912.04862 [cs, math, stat].

[13] Amit Daniely, Roy Frostig, and Yoram Singer. "Toward Deeper Understanding of Neural Networks: The Power of Initialization and a Dual View on Expressivity". In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee et al. Vol. 29. Curran Associates, Inc., 2016. URL: https://proceedings.neurips.cc/paper_files/paper/2016/file/abea47ba24142ed16b7d8fbf2c740e0d-Paper.pdf.

[14] Li Deng. "The MNIST Database of Handwritten Digit Images for Machine Learning Research". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[15] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. https://archive.ics.uci.edu/ml.

[16] Yarin Gal and Zoubin Ghahramani. "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning". In: *arXiv:1506.02142 [cs, stat]* (Oct. 2016). arXiv: 1506.02142 [cs, stat].

[17] Daniel Gissin and Shai Shalev-Shwartz. *Discriminative Active Learning*. July 2019. arXiv: 1907.06347 [cs, stat].

[18] Xavier Glorot and Yoshua Bengio. "Understanding the Difficulty of Training Deep Feedforward Neural Networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, Mar. 2010, pp. 249–256.

[19] Arthur Gretton et al. "A Kernel Two-Sample Test". In: *Journal of Machine Learning Research* 13.25 (2012), pp. 723–773. ISSN: 1533-7928.

[20] Soufiane Hayou, Arnaud Doucet, and Judith Rousseau. *Exact Convergence Rates of the Neural Tangent Kernel in the Large Depth Limit*. May 2022. arXiv: 1905.13654 [cs, stat].

[21] Bobby He, Balaji Lakshminarayanan, and Yee Whye Teh. "Bayesian Deep Ensembles via the Neural Tangent Kernel". In: *arXiv:2007.05864 [cs, stat]* (Oct. 2020). arXiv: 2007.05864 [cs, stat].

[22] Kaiming He et al. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. Feb. 2015. arXiv: 1502.01852 [cs].

[23] Tao He et al. "Towards Better Uncertainty Sampling: Active Learning with Multiple Views for Deep Convolutional Neural Network". In: *2019 IEEE International Conference on Multimedia and Expo (ICME)*. Shanghai, China: IEEE, July 2019, pp. 1360–1365. ISBN: 978-1-5386-9552-4. DOI: 10.1109/ICME.2019.00236.

[24] Trong Nghia Hoang, Quang Minh Hoang, and Bryan Kian Hsiang Low. "A Unifying Framework of Anytime Sparse Gaussian Process Regression Models with Stochastic Variational Inference for Big Data". In: *Proceedings of the 32nd International Conference on Machine Learning*. PMLR, June 2015, pp. 569–578.

[25] Neil Houlsby et al. *Bayesian Active Learning for Classification and Preference Learning*. Dec. 2011. arXiv: 1112.5745 [cs, stat].

[26] Arthur Jacot, Franck Gabriel, and Clément Hongler. "Neural Tangent Kernel: Convergence and Generalization in Neural Networks". In: *arXiv:1806.07572 [cs, math, stat]* (Feb. 2020). arXiv: 1806.07572 [cs, math, stat].

[27] Parnian Kassraie and Andreas Krause. "Neural Contextual Bandits without Regret". In: *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*. PMLR, May 2022, pp. 240–278.

[28] Kwanyoung Kim et al. *Task-Aware Variational Adversarial Active Learning*. Dec. 2020. arXiv: 2002.04709 [cs, stat].

[29] Andreas Kirsch, Tom Rainforth, and Yarin Gal. *Test Distribution-Aware Active Learning: A Principled Approach Against Distribution Shift and Outliers*. Nov. 2021. arXiv: 2106.11719 [cs, stat].

[30] Andreas Kirsch, Joost van Amersfoort, and Yarin Gal. "BatchBALD: Efficient and Diverse Batch Acquisition for Deep Bayesian Active Learning". In: *arXiv:1906.08158 [cs, stat]* (Oct. 2019). arXiv: 1906.08158 [cs, stat].

[31] Pang Wei Koh and Percy Liang. "Understanding Black-box Predictions via Influence Functions". In: *arXiv:1703.04730 [cs, stat]* (Dec. 2020). arXiv: 1703.04730 [cs, stat].

[32] Andreas Krause and Carlos Guestrin. "Nonmyopic Active Learning of Gaussian Processes: An Exploration-Exploitation Approach". In: *Proceedings of the 24th International Conference on Machine Learning*. ICML '07. New York, NY, USA: Association for Computing Machinery, June 2007, pp. 449–456. ISBN: 978-1-59593-793-3. DOI: 10.1145/1273496.1273553.

[33] Andreas Krause, Ajit Singh, and Carlos Guestrin. "Near-Optimal Sensor Placements in Gaussian Processes: Theory, Efficient Algorithms and Empirical Studies". In: *The Journal of Machine Learning Research* 9 (June 2008), pp. 235–284. ISSN: 1532-4435.

[34] Alex Krizhevsky. "Learning Multiple Layers of Features from Tiny Images". en. In: ().

[35] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. "Simple and Scalable Predictive Uncertainty Estimation Using Deep Ensembles". In: *arXiv:1612.01474 [cs, stat]* (Nov. 2017). arXiv: 1612.01474 [cs, stat].

[36] Jaehoon Lee et al. "Wide Neural Networks of Any Depth Evolve as Linear Models Under Gradient Descent". In: *Journal of Statistical Mechanics: Theory and Experiment* 2020.12 (Dec. 2020), p. 124002. ISSN: 1742-5468. DOI: 10.1088/1742-5468/abc62b. arXiv: 1902.06720.

[37] Chaoyue Liu, Libin Zhu, and Mikhail Belkin. *On the Linearity of Large Non-Linear Models: When and Why the Tangent Kernel Is Constant*. Feb. 2021. arXiv: 2010.01092 [cs, stat].

[38] David J. C. MacKay. "A Practical Bayesian Framework for Backpropagation Networks". In: *Neural Computation* 4.3 (1992), pp. 448–472. DOI: 10.1162/neco.1992.4.3.448.

[39] David J. C. MacKay. "The Evidence Framework Applied to Classification Networks". In: *Neural Computation* 4.5 (Sept. 1992), pp. 720–736. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/neco.1992.4.5.720.

[40] Michel Minoux. "Accelerated greedy algorithms for maximizing submodular set functions". In: *Optimization Techniques*. Ed. by J. Stoer. Berlin, Heidelberg: Springer Berlin Heidelberg, 1978, pp. 234–243. ISBN: 978-3-540-35890-9.

[41] Baharan Mirzasoleiman et al. *Lazier Than Lazy Greedy*. arXiv:1409.7938 [cs]. Nov. 2014. URL: http://arxiv.org/abs/1409.7938 (visited on 01/25/2023).

[42] Mohamad Amin Mohamadi, Wonho Bae, and Danica J. Sutherland. "Making Look-Ahead Active Learning Strategies Feasible with Neural Tangent Kernels". In: (2022). DOI: 10.48550/arXiv.2206.12569. arXiv: 2206.12569 [cs, stat].

[43] Brady Neal et al. *A Modern Take on the Bias-Variance Tradeoff in Neural Networks*. Dec. 2019. arXiv: 1810.08591 [cs, stat].

[44] Radford M Neal. *Bayesian learning for neural networks*. Vol. 118. Springer Science & Business Media, 2012.

[45] Radford M. Neal. *Bayesian Learning for Neural Networks*. Ed. by P. Bickel et al. Vol. 118. Lecture Notes in Statistics. New York, NY: Springer New York, 1996. ISBN: 978-0-387-94724-2 978-1-4612-0745-0. DOI: 10.1007/978-1-4612-0745-0.

[46] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. "An Analysis of Approximations for Maximizing Submodular Set Functions—I". In: *Mathematical Programming* 14.1 (Dec. 1978), pp. 265–294. ISSN: 0025-5610, 1436-4646. DOI: 10.1007/BF01588971.

[47] Yuval Netzer et al. "Reading Digits in Natural Images with Unsupervised Feature Learning". en. In: ().

[48] Roman Novak et al. *Neural Tangents: Fast and Easy Infinite Neural Networks in Python*. Dec. 2019. arXiv: 1912.02803 [cs, stat].

[49] Tim Pearce, Alexandra Brintrup, and Jun Zhu. *Understanding Softmax Confidence and Uncertainty*. 2021. arXiv: 2106.04972 [cs.LG].

[50] Viraj Prabhu et al. *Active Domain Adaptation via Clustering Uncertainty-weighted Embeddings*. Oct. 2021. arXiv: 2010.08666 [cs].

[51] Joaquin Quiñonero-Candela and Carl Edward Rasmussen. "A Unifying View of Sparse Approximate Gaussian Process Regression". In: *Journal of Machine Learning Research* 6.65 (2005), pp. 1939–1959. ISSN: 1533-7928.

[52] Hiranmayi Ranganathan et al. "Deep Active Learning for Image Classification". In: *2017 IEEE International Conference on Image Processing (ICIP)*. Beijing: IEEE, Sept. 2017, pp. 3934–3938. ISBN: 978-1-5090-2175-8. DOI: 10.1109/ICIP.2017.8297020.

[53] Carl Edward Rasmussen and Joaquin Quiñonero-Candela. "Healing the Relevance Vector Machine through Augmentation". In: *Proceedings of the 22nd International Conference on Machine Learning*. ICML '05. Bonn, Germany: Association for Computing Machinery, 2005, pp. 689–696. ISBN: 1595931805. DOI: 10.1145/1102351.1102438. URL: https://doi.org/10.1145/1102351.1102438.

[54] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. Cambridge, Mass: MIT Press, 2006. ISBN: 978-0-262-18253-9.

[55] Pengzhen Ren et al. "A Survey of Deep Active Learning". In: *arXiv:2009.00236 [cs, stat]* (Dec. 2021). arXiv: 2009.00236 [cs, stat].

[56] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2017. arXiv: 1609.04747 [cs.LG].

[57] Matthias W. Seeger, Christopher K. I. Williams, and Neil D. Lawrence. "Fast Forward Selection to Speed Up Sparse Gaussian Process Regression". In: *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*. Ed. by Christopher M. Bishop and Brendan J. Frey. Vol. R4. Proceedings of Machine Learning Research. Reissued by PMLR on 01 April 2021. PMLR, Mar. 2003, pp. 254–261. URL: https://proceedings.mlr.press/r4/seeger03a.html.

[58] Ozan Sener and Silvio Savarese. "Active Learning for Convolutional Neural Networks: A Core-Set Approach". In: *arXiv:1708.00489 [cs, stat]* (June 2018). arXiv: 1708.00489 [cs, stat].

[59] Burr Settles. *Active Learning*. Cham: Springer International Publishing, 2012. ISBN: 978-3-031-00432-2 978-3-031-01560-1. DOI: 10.1007/978-3-031-01560-1.

[60] Yao Shu et al. "NASI: LABEL- AND DATA-AGNOSTIC NEURAL ARCHITECTURE SEARCH AT INITIALIZATION". In: (2022), p. 25.

[61] Changjian Shui et al. "Deep Active Learning: Unified and Principled Method for Query and Training". In: *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. PMLR, June 2020, pp. 1308–1318.

[62] Samarth Sinha, Sayna Ebrahimi, and Trevor Darrell. *Variational Adversarial Active Learning*. Oct. 2019. arXiv: 1904.00370 [cs, stat].

[63] Alex Smola and Peter Bartlett. "Sparse Greedy Gaussian Process Regression". In: *Advances in Neural Information Processing Systems*. Vol. 13. MIT Press, 2000.

[64] Edward Snelson and Zoubin Ghahramani. "Local and Global Sparse Gaussian Process Approximations". In: *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*. PMLR, Mar. 2007, pp. 524–531.

[65] Mattias Teye, Hossein Azizpour, and Kevin Smith. *Bayesian Uncertainty Estimation for Batch Normalized Deep Networks*. July 2018. arXiv: 1802.06455 [stat].

[66] Antonio Torralba, Rob Fergus, and William T. Freeman. "80 Million Tiny Images: A Large Data Set for Nonparametric Object and Scene Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30.11 (2008), pp. 1958–1970. DOI: 10.1109/TPAMI.2008.128.

[67] Sattar Vakili et al. "Optimal Order Simple Regret for Gaussian Process Bandits". In: *arXiv:2108.09262 [cs, stat]* (Aug. 2021). arXiv: 2108.09262 [cs, stat].

[68] Sattar Vakili et al. "Uniform Generalization Bounds for Overparameterized Neural Networks". In: *arXiv:2109.06099 [cs, stat]* (Oct. 2021). arXiv: 2109.06099 [cs, stat].

[69] Dan Wang and Yi Shang. "A New Active Labeling Method for Deep Learning". In: *2014 International Joint Conference on Neural Networks (IJCNN)*. Beijing, China: IEEE, July 2014, pp. 112–119. ISBN: 978-1-4799-1484-5 978-1-4799-6627-1. DOI: 10.1109/IJCNN.2014.6889457.

[70] Zhilei Wang et al. "Neural Active Learning with Performance Guarantees". In: *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., 2021, pp. 7510–7521.

[71] Zhaoxuan Wu, Yao Shu, and Bryan Kian Hsiang Low. "DAVINZ: Data Valuation Using Deep Neural Networks at Initialization". In: *Proceedings of the 39th International Conference on Machine Learning*. PMLR, June 2022, pp. 24150–24176.

[72] Sergey Zagoruyko and Nikos Komodakis. *Wide Residual Networks*. June 14, 2017. arXiv: 1605.07146[cs]. URL: http://arxiv.org/abs/1605.07146 (visited on 01/22/2023).

[73] Beichen Zhang et al. *State-Relabeling Adversarial Active Learning*. Apr. 2020. arXiv: 2004.04943 [cs].

Appendix

Appendix A

Experimental Setup

A.1 Regression Experiments

We will use a combination of generated dataset and real-life data. Randomly generated data, referred to as Random Model, is constructed by labeling random points in a ball using a random model initialization.

Meanwhile the real-life training datasets are taken from the UCI Machine Learning Repository [15]. For all datasets, we split the whole data in half, and let one half be the pool of unlabeled data and the other be the test data which the algorithm has no access to. All the datasets used are regression datasets, with the exception of the Handwritten Digits dataset which is a classification dataset, but we perform regression on the label value (i.e. for the inputs corresponding to the digit n, we assign $y = n$, $\forall n \in \{1, \ldots, 10\}$). All datasets are also normalized such that they have mean 0 and variance of 1.

In all the regression experiments, the model used is a 2-layer multilayer perceptron (MLP) with hidden layer widths 512 and with bias. We set $\sigma_W = 1.$ and $\sigma_b = 0.1$. The NNs are optimized using gradient descent with step size 0.01.

A.2 Classification Experiments

For the classification experiments, in order to reduce the training time, we restrict the unlabeled data pool to a random subset of the whole training set. For the MNIST dataset (Section B.1), we randomly select $10,000$ points and use it as our unlabeled pool, while for the remaining classification experiments we randomly select $20,000$ points for the unlabeled pool. All the inputs are also rescaled so that the input values are between $[-1, 1]$. For all the models, we train the models using stochastic gradient descent with learning rate of 0.1 and decay rate of 0.005. The models are trained with training batch size of 32 and are trained for 100 epochs. Below, we provide a brief description of the datasets used and the model architectures used for training on the corresponding datasets.

Appendix B

Datasets

B.1 MNIST

The Modified National Institute of Standards and Technology (MNIST) dataset [14] is a widely-used collection of handwritten digit images, serving as a fundamental benchmark for various machine learning and computer vision algorithms.

The dataset comprises a total of $70,000$ grayscale images, with each image featuring a single handwritten digit (0-9). The images are of size 28×28 pixels, with pixel values ranging from 0 (black) to 255 (white). The dataset is divided into two subsets: a training set containing $60,000$ images and a testing set consisting of $10,000$ images.

MNIST images are derived from the original NIST dataset, which contains handwritten digits from American Census Bureau employees and high school students. The MNIST dataset is a cleaned and standardized version, with images centered and size-normalized to fit within the 28×28 pixel grid. These preprocessing steps minimize variations and reduce computational complexity, making the dataset suitable for machine learning tasks.

B.2 EMNIST

The Extended Modified National Institute of Standards and Technology (EMNIST) dataset [11] is a more comprehensive collection of handwritten characters, building upon the original MNIST dataset. It expands the scope of the widely-used MNIST dataset to include both digits and alphabetic characters, thereby offering a more challenging benchmark for machine learning and computer vision algorithms.

The EMNIST dataset consists of $814,255$ images, each sized 28×28 pixels with pixel values ranging from 0 (black) to 255 (white). These images contain handwritten digits (0-9) and alphabetic characters (A-Z, a-z), sourced from the original NIST dataset's Special Database 19. The EMNIST dataset is split into six subsets, catering to different application scenarios:

1. EMNIST ByClass: Comprising 62 classes (10 digits, 26 uppercase letters, and 26 lowercase letters), this set contains $814,255$ images, with $697,932$ images for training and $116,323$ images for testing.

2. EMNIST ByMerge: Featuring 47 classes, this set merges some uppercase and lowercase letters with similar shapes.

3. EMNIST Balanced: Consisting of 47 classes, this set aims to balance the number of images per class. The dataset contains $112,800$ images, with $94,000$ images ($2,000$ per class) designated for training and $18,800$ images (400 per class) for testing.

4. EMNIST Letters: Containing 26 classes of uppercase and lowercase letters, this set has $145,600$ images, with $124,800$ images for training and $20,800$ images for testing.

5. EMNIST Digits: Similar to the original MNIST dataset, this set comprises 10 classes of digits and consists of $280,000$ images, with $240,000$ images reserved for training and $40,000$ images for testing.

6. EMNIST MNIST: A direct replacement for the original MNIST dataset, this set has undergone the same preprocessing and balancing as the MNIST dataset, with $70,000$ images, $60,000$ for training, and $10,000$ for testing.

B.3 SVHN

The Street View House Numbers (SVHN) dataset [47] is a large-scale, real-world dataset of house number images, extracted from Google Street View images. The SVHN dataset provides a complex and diverse benchmark for machine learning and computer vision algorithms, particularly those focused on digit recognition and localization tasks.

The SVHN dataset comprises over $600,000$ 32×32 pixel color images of house numbers. These images are collected from a wide variety of geographic locations and feature diverse architectural styles, fonts, colors, and orientations. The dataset is divided into three subsets:

1. SVHN Train: Containing $73,257$ images, this set is intended for training machine learning algorithms.

2. SVHN Test: Comprising $26,032$ images, this set is used for evaluating the performance of trained models on previously unseen data.

3. SVHN Extra: With $531,131$ images, this set provides additional, less difficult samples that can be used for training or validation purposes.

Each image in the dataset is accompanied by a bounding box, which indicates the location of the house number in the image, and a label that specifies the digit sequence. The images in the SVHN dataset are more challenging than those in the MNIST or EMNIST datasets due to factors such as variable lighting conditions, occlusions, and distortions.

B.4 CIFAR-100

The CIFAR-100 dataset [34] is a widely-used collection of color images, intended for object recognition and classification tasks in the field of machine learning and computer vision. The CIFAR-100 dataset serves as a more challenging benchmark compared to its counterpart, the CIFAR-10 dataset, due to the increased number of classes and finer classification categories.

The CIFAR-100 dataset comprises $60,000$ color images, each of size 32×32 pixels. These images are divided into 100 classes, representing different objects, animals, and scenes. Each class contains 600 images, with 500 designated for training and the remaining 100 for testing, resulting in a total of $50,000$ training images and $10,000$ testing images. The dataset is further divided into 20 *superclasses*, with each superclass containing five related classes, providing the opportunity to explore both coarse and fine-grained classification tasks.

Images in the CIFAR-100 dataset are sourced from the Tiny Images dataset [66], a collection of 80 million small images obtained from the internet. The images in the CIFAR-100 dataset have been downsampled and preprocessed, which includes resizing, center-cropping, and normalizing pixel values.

Appendix C

Active Learning Baselines

C.1 Random

In the random selection strategy for neural active learning, a predetermined number of unlabeled samples are randomly selected from the pool of available data without considering any specific criteria. It is a straightforward strategy that serves as a baseline for comparison with other more sophisticated strategies.

Although the random selection strategy lacks the sophistication of other active learning strategies, it offers some advantages:

1. Simplicity: The random selection strategy is easy to implement and requires minimal computational overhead, making it suitable for cases where more complex strategies are not feasible or as a baseline for comparison.

2. Unbiased: Random selection does not introduce any bias toward specific samples or regions in the feature space, ensuring that the selected samples provide a fair representation of the entire dataset.

3. Serendipity: In some cases, random selection may accidentally pick highly informative samples that other strategies might overlook due to their focus on specific criteria.

Despite these advantages, the random selection strategy is generally considered less effective than other active learning strategies as it does not specifically target samples that would provide the most significant improvement in the model's performance. In most cases, it is advisable to use more advanced strategies that can intelligently prioritize samples based on their potential impact on the model's learning.

C.2 K-Means++

K-means++ is an algorithm for choosing the initial centroids in the K-means clustering process, aiming to improve the convergence speed and the final clustering quality [3]. It addresses the shortcomings of the standard K-means algorithm, which is sensitive to the

initial placement of centroids and can result in poor convergence or getting stuck in local optima. The K-means++ algorithm can be employed as a data selection strategy to identify the most representative and diverse samples from an unlabeled dataset. By adapting the K-means++ algorithm to select samples for labeling, we can ensure that the chosen instances cover a broad range of the feature space, allowing the model to learn from a more informative and diverse set of examples. The K-Means++ algorithm works as follows:

1. Set the desired number of samples (k) to be selected for labeling.
2. Choose the first sample randomly from the pool of unlabeled data.
3. Calculate the distance between each remaining unlabeled sample and the selected sample(s).
4. Select the next sample from the pool with a probability proportional to the square of its distance from the nearest selected sample. This step ensures that samples farther away from the already selected samples are more likely to be chosen, promoting diversity in the selected set.
5. Repeat steps 3 and 4 until k samples are selected for labeling.
6. Request the oracle to label the selected samples, and add them to the labeled dataset.
7. Train or fine-tune the model on the updated labeled dataset.
8. Repeat the process iteratively until a stopping criterion is met, such as a maximum number of iterations or a desired performance threshold.

When using this algorithm, all the points are selected right from the start (and order in selection by the algorithm is kept). When the user queries for a batch of size b, the algorithm returns the next b elements that it has chosen from the K-Means++ initialization algorithm.

C.3 BADGE

BADGE (Batch Active learning by Diverse Gradient Embeddings) [4] is designed for selecting a diverse and informative batch of samples from an unlabeled dataset to enhance the learning process of a model. BADGE utilizes the hallucinated gradient space, or the gradient of the loss function with respect to the final output layer,

$$h(x) = \frac{\partial}{\partial \theta^{(-1)}} \mathcal{L}((x, \tilde{y}), \theta)$$

where $\theta^{(-1)}$ are the model parameters of the final layer and $\tilde{y}$ is the pseudo-prediction from the model based on the model output (for example, in classification problems, $\tilde{y}$ would be the one-hot vector representing the class prediction given by the model output). The BADGE algorithm works as follows:

1. Train the model on the current labeled dataset.

2. Calculate the loss gradients for each unlabeled sample with respect to the model's parameters. These gradients represent how much the model's parameters would change if the sample were included in the training set.

3. Embed the high-dimensional gradients of each sample into a lower-dimensional space using dimensionality reduction techniques, such as PCA (Principal Component Analysis) or t-SNE (t-Distributed Stochastic Neighbor Embedding). This step reduces the computational complexity of the algorithm while preserving the essential structure of the gradients.

4. Apply a diversity-promoting algorithm, such as K-means++, on the lower-dimensional gradient embeddings to select a batch of samples that are both diverse and informative. This step ensures that the chosen samples cover a wide range of gradient directions, which is crucial for improving the model's generalization capabilities.

5. Request the oracle to label the selected samples and add them to the labeled dataset.

6. Train or fine-tune the model on the updated labeled dataset.

7. Repeat the process iteratively until a stopping criterion is met, such as a maximum number of iterations or a desired performance threshold.

Theoretically, BADGE is able to provide a balance between selecting points which are diverse and selecting points which the model is uncertain about. Unlike in the original paper, for experiments involving BADGE, we do not provide the algorithm with an initial training set.

C.4 MLMOC

The Most Likely Model Output Change (MLMOC) algorithm [42] focuses on selecting samples with the highest expected change in model output. This approach aims to reduce the overall uncertainty in the model's predictions by targeting samples that are most likely to influence the model's decisions. The MLMOC algorithm is particularly useful when dealing with complex learning tasks, as it can efficiently select informative samples that lead to improved model performance with fewer labeled instances. The MLMOC algorithm works as follows:

1. Train the model on the current labeled dataset.

2. For each unlabeled sample in the dataset, compute the model's output (e.g., class probabilities for classification tasks) and identify the model's current prediction (i.e., the class with the highest probability).

3. Estimate the change in model output for each sample by perturbing the model's parameters, simulating the effect of incorporating the sample into the training set. This step can be done using various techniques, such as the gradient of the model's output with respect to its parameters, or using a surrogate model to approximate the model's behavior.

4. Calculate the expected change in model output for each sample by integrating over all possible true labels, weighted by the model's uncertainty about the true label. This step can be achieved using the model's class probabilities or other measures of uncertainty.

5. Select the sample(s) with the highest expected change in model output for expert labeling.

6. Request the oracle to label the selected samples and add them to the labeled dataset.

7. Train or fine-tune the model on the updated labeled dataset.

8. Repeat the process iteratively until a stopping criterion is met, such as a maximum number of iterations or a desired performance threshold.

The main advantage of the MLMOC active learning algorithm is its ability to identify informative samples that are most likely to influence the model's decisions, leading to improved model performance with fewer labeled instances. By focusing on the expected change in model output, the MLMOC algorithm reduces the overall uncertainty in the model's predictions, resulting in more accurate and confident decisions.

Appendix D

Reported Metrics

To quantify the model performances, we use either the test mean-squared error (MSE) for regression problems, or the test accuracy for classification problems. To quantify the initialization robustness of the models, we use *output variance* for regression tasks and *output entropy* for classification tasks.

D.1 Output Variance

Output variance is used as an initialization-robustness measure for regression tasks, and is the empirical variance of output of trained neural networks with respect to the different model initializations. In our experiments, we report the 90^{th} percentile output variance, which is defined as the 90th percentile output variance test data. The 90^{th} percentile is used instead of the 100^{th} percentile (i.e. the maximum output variance) since using the 100^{th} percentile tends to focus on some outlier rather than reflecting the output variance of the majority of the test data. We also chose to report the 90^{th} percentile value instead of the average value since it is able to indicate the output variance of the worst-case inputs and is therefore a better measure of initialization robustness on the overall space.

D.2 Output Entropy

Output entropy is used as an initialization-robustness measure for classification tasks, and is defined as the empirical entropy of the predicted label. For a neural network, the predicted label for some input x is given by $\hat{y} = \arg\max_i f(x;\theta)_i$. Given multiple trained models with different parameters $\theta_1, \ldots, \theta_k$, we will obtain different predictions $\hat{y}_1, \ldots, \hat{y}_k$. The output entropy is then defined as $-\sum_{i=1}^{c} v_i \log v_i$ where $v_i = \frac{1}{c} \cdot \sum_{j=1}^{k} \mathbb{1}_{\hat{y}_j = i}$. A lower output entropy corresponds to models whose predictions are more consistent with each other.

Appendix E

Computation of the NTK

E.1 Theoretical NTK using Neural-Tangents

The theoretical NTK is computed using the Neural-Tangents package [48]. Since different packages are used for AL and for model training, the trained network will not have exactly the same parameters as those used to compute the theoretical NTK. Regardless, we still find that the kernels themselves shows enough agreement and are still useful for predicting the network dynamics.

The main disadvantage of using the theoretical NTK is that it can not be computed for all model architectures. In particular, Neural-Tangents only supports AveragePooling layers but not MaxPooling layers. It also does not support BatchNorm layers or Dropout layers. For this reason, experiments involving the theoretical NTK are restricted to use the MLP networks. Also, we do not use the theoretical NTK for MLMOC, which uses the empirical kernel.

E.2 Empirical NTK Using PyTorch

An alternative method of computing the NTK is to do so empirically by taking the inner product of the model output gradient $\nabla_\theta f\left(x, \mathcal{X}\right)$ with respect to its parameters. This requires us to compute the Jacobian $\nabla_\theta f(x, \mathcal{X})$, which is extremely expensive in terms of required memory and cost for performing the matrix multiplication. In particular, for a model with p parameters and o outputs, computing the NTK on an input of size n requires $\mathcal{O}(npo)$ space and $\mathcal{O}(n^2po)$ running time for the matrix multiplication alone. In order to perform this operation efficiently, we follow the method as used in PyTorch's FuncTools tutorial[1]. Note that FuncTools is not compatible with all neural network components (e.g. BatchNorm), and therefore in our experiments we do not use those components in the models. Even though Neural-Tangents is able to compute the empirical NTK as well, we chose to compute the empirical NTK on PyTorch since the models and (most of) the model training are done on PyTorch anyway.

[1] See `https://pytorch.org/functorch/stable/notebooks/neural_tangent_kernels.html`.

For classification instances where the model has multiple outputs, we also perform a further approximation of only using the gradient which contributes to a single model output (which is chosen at random). This is possible since the gradient inner product with respect to each outputs are independent of each other and has the same value in the limit. This is a similar trick which is also used by [42].