Making Look-Ahead Active Learning Strategies Feasible with Neural Tangent Kernels

Mohamad Amin Mohamadi*
University of British Columbia
lemohama@cs.ubc.ca

Wonho Bae*
University of British Columbia
whbae@cs.ubc.ca

Danica J. Sutherland
University of British Columbia
Alberta Machine Intelligence Institute
dsuth@cs.ubc.ca

Abstract

We propose a new method for approximating active learning acquisition strategies that are based on retraining with hypothetically-labeled candidate data points. Although this is usually infeasible with deep networks, we use the neural tangent kernel to approximate the result of retraining, and prove that this approximation works asymptotically even in an active learning setup – approximating "look-ahead" selection criteria with far less computation required. This also enables us to conduct sequential active learning, i.e. updating the model in a streaming regime, without needing to retrain the model with SGD after adding each new data point. Moreover, our querying strategy, which better understands how the model's predictions will change by adding new data points in comparison to the standard ("myopic") criteria, beats other look-ahead strategies by large margins, and achieves equal or better performance compared to state-of-the-art methods on several benchmark datasets in pool-based active learning.

1 Introduction

Deep learning has drastically advanced over the past decade along with a dramatic increase in the volume of available data. It is, however, time-consuming and expensive to manually annotate large datasets. Active learning attempts to alleviate this by allowing a model to "actively" request annotation of specific data points, with the expectation that a model trained on informative points will learn a better prediction than one on a random, "passively" labeled set of the same size.

The most common form of active learning is based on using an *acquisition function* to measure the informativeness of potential data points or batches from some unlabeled pool. Many successful acquisition functions are based on the current model's uncertainty, such as maximum entropy [1] and Bayesian Active Learning by Disagreement (BALD) [2]. These functions are based on the expectation that training on points about which the current model is uncertain will be effective at decreasing the future model's uncertainty about similar points. This assumption, however, often does not hold: uncertain points might be inherently difficult to predict (aleatoric uncertainty) or simply too difficult for the model to learn right now.

It would be helpful, then, to see how a new data point would change a model, potentially with respect to other unseen data. One such strategy is known as Expected Model Change, which approximates

*These authors contributed equally.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

how much a model's parameters will change when observing a new hypothetically-labeled data point [3, 4]. This approach, though, only considers the magnitude of change in parameters, not the model's actual outputs on the data distribution, and it only looks at the size of the first stochastic gradient (SGD) step with the new data point rather than considering the full trajectory of training.

We will refer to acquisition strategies which consider full retraining of a model based on hypothetical observations as *look-ahead* criteria. Previous attempts to use look-ahead criteria based on the change in model output include Expected Error Reduction [5] and Expected Model Output Change [6]. These approaches have been used only with specialized models such as Naïve Bayes and Gaussian processes, where retraining on every new candidate data point is feasible. These classes of models, however, tend to not work as well as modern neural networks, meaning these approaches are outperformed by simpler acquisition functions on stronger models.

In this work, we propose an algorithm that makes it feasible to conduct active learning with look-ahead acquisition strategies using general neural networks. More specifically, we approximate a network using its Neural Tangent Kernel (NTK) [7], which makes it possible to obtain the behavior of retraining a network on a candidate data point without actually retraining the network. At each query step, we use the local approximation of the neural network as a proxy to compute a look-ahead acquisition function, from which we query a new data point. We prove that in the asymptotic regime where the width of the neural network goes to infinity, this approximation agrees with the result of actually retraining the network, even in an iterative setup such as in active learning. Our approximation decreases the wall-clock time more than 100 times compared to naïve computation of a look-ahead acquisition function, making them far more feasible in practice.

Our proposed method outperforms existing look-ahead strategies by large margins, and achieves equal or better performance compared to the state-of-the-art methods on several benchmark datasets in pool-based active learning, including MNIST [8], SVHN [9], CIFAR10 [10], and CIFAR100 [10]. The NTK approximation also makes it possible to decouple receiving new data labels and SGD training, unlike previous active learning methods, where knowing true labels does not add any information without SGD training. This is useful when annotation is fast (yet expensive) but model training is slow, since adding data to an existing NTK approximation is far faster than retraining with SGD. Sequentially adding true labels of new data gives performance substantially better than more common batch setups.

2 Preliminaries

Active learning. In pool-based active learning, we have a labeled set $\mathcal{L} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{|\mathcal{L}|}$ and unlabeled set $\mathcal{U} = \{x^{(i)}\}_{i=1}^{|\mathcal{U}|}$, where $x^{(i)}$s are inputs and $y^{(i)}$s are corresponding labels. At each time step t, a model f parameterized by θ is trained on the labeled set $\mathcal{L}$ which we denoted as $f_{\mathcal{L}}$, and then a data point from the unlabeled set $\mathcal{U}$ is selected to be labeled, according to an acquisition function A measuring the "information gain" of a potential data point x:

$$x^* = \operatorname*{argmax}_{x \in \mathcal{U}} A(x, f_{\mathcal{L}}, \mathcal{L}, \mathcal{U}). \tag{1}$$

One of the most common choices of the acquisition function is *maximum entropy*, which computes the estimated entropy of the unknown label of x, denoted as Y: $A_{\textit{entropy}}(x, f_{\mathcal{L}}, \mathcal{L}, \mathcal{U}) := \mathcal{H}(Y \mid x;\, f_{\mathcal{L}})$. Although the maximum entropy acquisition function often works well in practice, it does not consider how the model would change with a new queried data point, and so can overly prioritize inherently difficult, uninformative points (*e.g.* outliers).

Expected gradient length [11] approximates the *expected model change* by how large the gradient of the loss becomes when adding that data point, $\mathbb{E}_{y \sim p_\theta(\cdot|x)} \|\nabla \ell_\theta(\mathcal{L} \cup (x, y))\|$. BADGE [4], a more recent variant, lower-bounds the gradient norm of the last layer induced by any possible label. These approaches do not consider how the change in a model actually interacts with the data distribution: large parameter changes might give relatively small changes in predictions on most data points. They also consider a single SGD update, only roughly approximating total change over training.

One approach to alleviate this limitation is *expected error reduction* [EER; 5], given by

$$A_{\textit{EER}}(x, f_{\mathcal{L}}, \mathcal{L}, \mathcal{U}) := -\,\mathbb{E}_{y \sim p_\theta(\cdot|x)} \sum_{i=1}^{|\mathcal{U}|} \mathcal{H}(Y^{(i)} \mid x^{(i)};\, f_{\mathcal{L}^+}). \tag{2}$$

Here $f_{\mathcal{L}^+}$ refers to the model trained on $\mathcal{L} \cup \{(x, y)\}$, and $\mathcal{H}(Y^{(i)}|x^{(i)}; f_{\mathcal{L}^+})$ is the (estimated) entropy of the unknown label of $x^{(i)}$ in $\mathcal{U}$ using that model. Maximizing A_{EER} chooses the candidate point whose label has maximal mutual information to the labels of unlabeled data, $\mathcal{I}(Y; y| f_{\mathcal{L}})$. If $f_{\mathcal{L}}$ is a fairly good model and the problem is fairly difficult, the numerical value of A_{EER} is likely dominated by the sum of irreducible (aleatoric) uncertanties over the data points; it thus might be prone to noise in estimation based on those large terms.

Freytag et al. [6] propose instead *expected model output change* (EMOC), based on the difference in model predictions as measured by a distance $\mathcal{D}$ (*e.g.* the Euclidean distance between probability vectors). The acquisition function is defined as

$$A_{EMOC}(x, f_{\mathcal{L}}, \mathcal{L}, \mathcal{U}) := \mathbb{E}_{y \sim p_\theta(\cdot|x)} \sum_{i=1}^{|\mathcal{U}|} \mathcal{D}(f_{\mathcal{L}}(x^{(i)}), f_{\mathcal{L}^+}(x^{(i)})). \tag{3}$$

Intuitively, we might hope that the change in outputs roughly cancels out the irreducible (aleatoric) uncertainty, leaving us to focus primarily on the decrease in model (epistemic) uncertainty [12]. Our techniques also apply to EER, but EMOC-like criteria performed better in our initial exploration.

Despite their advantages, look-ahead acquisition functions have been in practical reach only with certain types of models: EER has been applied to Naïve Bayes [5] and Gaussian random fields [13], and EMOC only to Gaussian processess [14, 15]. Neural networks generally outperform those models, but obtaining $f_{\mathcal{L}^+}$ with SGD training for every candidate example in $\mathcal{U}$ is impractically expensive. In this work, we propose to approximate $f_{\mathcal{L}^+}$ with a more computationally efficient proxy based on neural tangent kernels, making look-ahead acquisition functions feasible on neural networks.

Neural tangent kernels. Over the past few years, connections between infinitely wide neural networks trained by gradient descent and kernel methods have become increasingly clear. The NNGP approximation of Matthews et al. [16], building off a line of work stemming from Neal [17], shows that if a network's parameters are initialized with an appropriate Gaussian distribution and only the last layer is trained, the resulting function agrees with a Gaussian process with the kernel $\mathcal{K}(x, x') = \mathbb{E}_\theta[f_\theta(x) f_\theta(x')]$, where the expectation is over initializations. Jacot et al. [7], building off several immediately preceding works on optimization of wide neural networks, show that infinitely wide fully-connected neural networks also follow Gaussian process behavior, with the kernel

$$\mathcal{K}(x, x') = \mathbb{E}_\theta \left[\left\langle \frac{\partial f_\theta(x)}{\partial \theta}, \frac{\partial f_\theta(x')}{\partial \theta} \right\rangle \right] \tag{4}$$

(derivatives here denoting Jacobians with respect to the vector of all parameters θ). The also show that this kernel remains constant over the course of training, so that these networks evolve like a linear model under kernel gradient descent. Among many important follow-ups, Arora et al. [18] expand the results to convolutional networks with finite but large widths, and Novak et al. [19] provide an implementation for nearly-arbitrary network structures. Yang [20] and Yang and Littwin [21] later showed that Jacot et al. [7]'s findings are architecturally universal, extending the domain from fully-connected networks to a large class of architectures including ResNets and Transformers.

Unfortunately, these infinite-width versions of networks tend not to generalize as well as finite networks trained by SGD. Hence, we still want to conduct active learning for finite neural networks, rather than for pure NTK models. The infinite-width NTK also tends to be a mediocre proxy for the behavior of training a finite neural network, and so using it as a proxy for $f_{\mathcal{L}^+}$ does not tend to work as well as we might hope (*e.g.* Figure 1b). We will thus instead use a local linearized approximation of the neural network, based on the empirical neural tangent kernel, as introduced next.

3 Active Learning using NTKs

In this section, we first define a neural network, and the linear model which (in the infinite-width limit) agrees with training that network. We will prove that in the infinite-width limit, sequentially retraining on increasing datasets – as in the active learning process – is the same as if we had trained from scratch on the final dataset. In practical regimes, however, the infinite-width NTK does not tend to agree with finite-width SGD very closely. We thus use a local linear approximation of the network, based on the empirical NTK, to approximate retraining in our look-ahead criteria.

Notation. We mostly use the same notation as Jacot et al. [7]. We define f as a fully-connected neural network with $L+1$ hidden layers numbered from 0 (input) to L (output), where each layer has $n_0, \ldots, n_L$ neurons. This network has number of parameters $P = \sum_{l=0}^{L-1}(n_l+1)n_{l+1}$: each layer has a weight matrix $W^{(l)} \in \mathbb{R}^{n_l \times n_{l+1}}$ and a bias vector $b^{(l)} \in \mathbb{R}^{n_{l+1}}$. The network is defined as f_θ, where θ collects all of the parameters: $\theta = \cup_{l=0}^{L}\theta^l$ and $\theta^l = \mathrm{vec}(W^{(l)}, b^{(l)})$. We denote a labeled set $\mathcal{L}$ as defined in Section 2 and use $\mathcal{X} = \{x : (x,y) \in \mathcal{L}\}$ and $\mathcal{Y} = \{y : (x,y) \in \mathcal{L}\}$ to denote its inputs and labels, respectively. We write $f_{\mathcal{D}_1} \xrightarrow[t=\infty]{\mathcal{D}_1 \cup \mathcal{D}_2} f_{\mathcal{D}_1 \cup \mathcal{D}_2}$ to mean that $f_{\mathcal{D}_1 \cup \mathcal{D}_2}$ is trained using gradient descent until convergence on the dataset $\mathcal{D}_2$, starting from $f_{\mathcal{D}_1}$.

Unless otherwise specified, the loss function associated with gradient descent training is assumed to be the squared loss, $\ell(\mathcal{Y}, f(\mathcal{X})) := \frac{1}{2}\|\mathcal{Y} - f(\mathcal{X}))\|_2^2$. This allows for far more efficient usage of NTKs, but Hui and Belkin [22] demonstrate that, with the right learning rate, L_2 loss is just as effective as cross-entropy for many vision and natural language processing tasks.

Neural networks in the infinite width limit. Jacot et al. [7] show that in the infinite width limit, when the network f is trained using gradient descent with the squared loss on a labeled set $\mathcal{L}$, the outputs of the network on any arbitrary point x evolve as

$$f_t(x) = f_0(x) + \mathcal{K}(x, \mathcal{X})\, \mathcal{K}(\mathcal{X}, \mathcal{X})^{-1}(\mathcal{I} - e^{-t\mathcal{K}(\mathcal{X},\mathcal{X})})(\mathcal{Y} - f_0(\mathcal{X})), \tag{5}$$

where $\mathcal{K}$ is the NTK of (4), which stays constant during training. Here we treat inputs $\mathcal{X}$ as a matrix and labels $\mathcal{Y}$ as a vector in corresponding order.

Consider sequentially training a network on increasing datasets $S_1 \subset S_2 \subset \cdots \subset S_C$, warm-starting each time from the result of training on the previous dataset. We build on the results of Jacot et al. [7] to show that the final network from this process is asymptotically equivalent, in the infinite-width limit, to training the same network from scratch (cold-start) on the last, biggest dataset S_C. This justifies our efficient approximation of the retrained network, even after we have gone through many iterations of active learning. See Appendix A for a formal statement and proof.

Theorem 3.1 (Informal)**.** *Let $S_1, S_2, \ldots, S_C$ be C datasets such that for all $i > j$, $S_j \subset S_i$. Let f_0 be a randomly initialized network. Let $f_{S_{1,2,\ldots,C}}$ be the network resulting from starting at f_0 and training f using gradient descent on datasets $S_1, \ldots, S_C$ sequentially until convergence:*

$$f_0 \xrightarrow[t=\infty]{S_1} f_{S_1} \xrightarrow[t=\infty]{S_2} f_{S_{1,2}} \cdots \xrightarrow[t=\infty]{S_C} f_{S_{1,2,\ldots,C}}.$$

Also, let f_{S_C} be $f_0 \xrightarrow[t=\infty]{S_C} f_{S_C}$. Assuming that f has $L+1$ layers such that taking $n_0, n_1, \ldots, n_L \to \infty$ sequentially, we have for any arbitrary data point x that $f_{S_C}(x) = f_{S_{1,2,\ldots,C}}(x)$.

NTK approximations of retrained networks. Armed with Theorem 3.1, we can now approximate the outputs of a neural network from retraining. Suppose we have a labeled set $\mathcal{L}$ and unlabeled set $\mathcal{U}$ as defined in Section 2. A neural network $f_\mathcal{L}$ (whose layers are assumed to be infinitely wide) has been trained on $\mathcal{L}$ as $f_0 \xrightarrow[t=\infty]{\mathcal{L}} f_\mathcal{L}$. We are interested in characterizing the outcome after retraining this neural network using each of the data points in $\mathcal{U}$. In other words, for each $x' \in \mathcal{U}$ with a hypothetical label y' and $\mathcal{L}^+ := \mathcal{L} \cup (x', y')$, we would like to know what $f_\mathcal{L} \xrightarrow[t=\infty]{\mathcal{L}^+} f_{\mathcal{L}^+}$ looks like.

According to (5), the outputs of $f_\mathcal{L}$ for an arbitrary data point x can be formulated as

$$f_\mathcal{L}(x) = f_0(x) + \mathcal{K}(x, \mathcal{X})\, \mathcal{K}(\mathcal{X}, \mathcal{X})^{-1}(\mathcal{Y} - f_0(\mathcal{X})). \tag{6}$$

Let $\mathcal{X}^+$ and $\mathcal{Y}^+$ be inputs and labels of $\mathcal{L}^+$. Then, based on Theorem 3.1, we can conclude that

$$f_{\mathcal{L}^+}(x) = f_0(x) + \mathcal{K}(x, \mathcal{X}^+)\mathcal{K}(\mathcal{X}^+, \mathcal{X}^+)^{-1}(\mathcal{Y}^+ - f_0(\mathcal{X}^+)). \tag{7}$$

As $\mathcal{K}$ remains constant in the infinite-width regime, most of the relevant quantities can be reused:

$$\mathcal{K}(x, \mathcal{X}^+) = \begin{bmatrix}\mathcal{K}(x, \mathcal{X}) & \mathcal{K}(x, x')\end{bmatrix}, \qquad \mathcal{K}(\mathcal{X}^+, \mathcal{X}^+) = \begin{bmatrix}\mathcal{K}(\mathcal{X}, \mathcal{X}) & \mathcal{K}(\mathcal{X}, x') \\ \mathcal{K}(x', \mathcal{X}) & \mathcal{K}(x', x')\end{bmatrix}. \tag{8}$$

Remark 3.2. *In the infinite width limit, we can analytically obtain the predictions of a neural network after retraining on additional data points by augmenting its corresponding NTK using (8).*

Several works have shown that while being an inspiring theoretical motivation, neural networks in the infinite width limit do not work as well as their finite-width counterparts [18, 23, 24]. Although Arora et al. [18] prove non-asymptotic bounds between these infinite width neural networks and their corresponding finite-width networks, empirical studies in [20, 25] have shown that this approximation may not be effective for practical network widths. (If it were, we wouldn't need a network at all, and would simply do all our learning with a Gaussian process based on the NTK.) Thus, we would like to be able to efficiently characterize the outputs of a finite neural network after retraining.

Lee et al. [25] showed the first-order Taylor expansion of a neural network around its initialization (a *linearized neural network*) has training dynamics converging to that of the neural network as the width grows. That is, let f_t denote the network which has been trained with gradient descent for t steps, and f_t^{lin} the result of training the linearized network for t steps. They prove that, under some regularity conditions and when the learning rate η is less than a certain threshold,

$$\sup_{t\geq 0} \|f_t(x) - f_t^{lin}(x)\|_2 = \sup_{t\geq 0} \|\Theta_t - \Theta_0\|_F = \mathcal{O}\left(\frac{1}{\sqrt{\text{width}}}\right). \tag{9}$$

Informally, when f is wide enough, its predictions can be approximated by those of the linearized network f^{lin}. This is attractive since f^{lin} has simple training dynamics, converging to

$$f_{\mathcal{L}}^{lin}(x) = f_0(x) + \Theta_0(x, \mathcal{X})\, \Theta_0(\mathcal{X}, \mathcal{X})^{-1}(\mathcal{Y} - f_0(\mathcal{X})), \tag{10}$$

where Θ_0 is the *empirical* tangent kernel of f_0, $\Theta_0(x, y) := \nabla_\theta f_0(x)\, \nabla_\theta f_0(y)^\top$.

In the infinite-width limit, the empirical NTK Θ_t converges to the NTK $\mathcal{K}$ almost surely, leading (10) to become equivalent to (6). In finite-width regimes, however, it is intuitive to expect that the *local* approximation may still be able to handle *local* retraining, such as $f_{\mathcal{L}} \xrightarrow[t=\infty]{\mathcal{L}\cup\{(x,y)\}} f_{\mathcal{L}\cup\{(x,y)\}}$, even when the correspondence to the infinite-width limit $\mathcal{K}$ is loose. We thus model retraining of a finite network by augmenting the empirical NTK as in (8), as justified by Theorem 3.1 and Remark 3.2. Specifically, we approximate the finite width network $f_{\mathcal{L}^+}(x)$ defined by $f_{\mathcal{L}} \xrightarrow[t=\infty]{\mathcal{L}^+} f_{\mathcal{L}^+}$ using

$$f_{\mathcal{L}^+}(x) \approx f_{\mathcal{L}^+}^{lin}(x) = f_{\mathcal{L}}(x) + \Theta_{\mathcal{L}}(x, \mathcal{X}^+)\Theta_{\mathcal{L}}(\mathcal{X}^+, \mathcal{X}^+)^{-1}\left(\mathcal{Y}^+ - f_{\mathcal{L}}(\mathcal{X}^+)\right) \tag{11}$$

where $\Theta_{\mathcal{L}}$ is the empirical NTK obtained from the parameters of $f_{\mathcal{L}}$.

Efficient computation. To avoid computing the full empirical NTK with the shape of $LC \times LC$ where L is the size of the labeled set and C is the number of classes, we use the "single-logit" approximation as explained in Wei et al. [26] (Section 2.3) in which we only compute the corresponding NTK of the first logit of the network, i.e. $\Theta(x, y) = \nabla_\theta f^{(1)}(x)^\top \nabla_\theta f^{(1)}(y) \otimes I_C$ where $f^{(1)}(x)$ refers to the first neuron of the output of f on the datapoint x. For a kernel regression, we can take the kronecker product out, which decreases the memory and time complexity of computing the NTK by an order of $\mathcal{O}(C^2)$.

To compute (11), we must solve linear systems with $\Theta_{\mathcal{L}}(\mathcal{X}^+, \mathcal{X}^+)$. This can also be done efficiently using the block structure of $\Theta_{\mathcal{L}}(\mathcal{X}^+, \mathcal{X}^+)$: letting $\mathbf{v} = \Theta_{\mathcal{L}}(\mathcal{X}, \mathcal{X})\Theta_{\mathcal{L}}(\mathcal{X}, x')$ and $u = \Theta_{\mathcal{L}}(x', x') - \Theta_{\mathcal{L}}(x', \mathcal{X})\Theta_{\mathcal{L}}(\mathcal{X}, \mathcal{X})^{-1}\Theta_{\mathcal{L}}(\mathcal{X}, x')$,

$$f_{\mathcal{L}^+}^{lin}(x) = f_{\mathcal{L}}^{lin}(x) + \frac{1}{u}\left(\Theta_{\mathcal{L}}(x, \mathcal{X})\mathbf{v} - \Theta_{\mathcal{L}}(x, x')\right)\left(\mathbf{v}^T(\mathcal{Y} - f_{\mathcal{L}}(\mathcal{X})) - (y' - f_{\mathcal{L}}(x'))\right). \tag{12}$$

Thus, rather than inverting $\Theta_{\mathcal{L}}(\mathcal{X}^+, \mathcal{X}^+)$ for each $x \in \mathcal{U}$, we can invert $\Theta_{\mathcal{L}}(\mathcal{X}, \mathcal{X})$ only once, then use some matrix multiplications to find $f_{\mathcal{L}^+}^{lin}(x)$ for each query point.

Time complexity. We now provide the time complexity of the NTK approximation proposed in Equation (11) (using efficient block computation in Equation (12)). Let L be the size of the labeled set, U of the unlabeled set, P the number of model parameters, and E the number of training epochs. The computation of $\Theta_{\mathcal{L}}(\mathcal{X}_{\mathcal{U}}, \mathcal{X}_{\mathcal{L}})$, $\Theta_{\mathcal{L}}(\mathcal{X}_{\mathcal{L}}, \mathcal{X}_{\mathcal{L}})$, and $\Theta_{\mathcal{L}}(\mathcal{X}_{\mathcal{L}}, \mathcal{X}_{\mathcal{L}})^{-1}$ take $\mathcal{O}(LUP)$, $\mathcal{O}(L^2P)$ and $\mathcal{O}(L^3)$ time, respectively. If we compute (12) for all $x \in \mathcal{U}$, the matrix multiplication takes $\mathcal{O}(UL^2)$. Putting altogether, the time complexity of the NTK approximation is $\mathcal{O}(LUP + L^2P + L^3 + UL^2)$. Since we assume $P > \max(L, U)$ for overparameterized models, the time complexity can be simplified to $\mathcal{O}(LUP + L^2P)$.

Algorithm 1: Active learning using NTKs

Input: Initialize a model f_0, labeled pool $\mathcal{L}_0$, and unlabeled pool $\mathcal{U}_0$
Train f_0 on the initial labeled set $\mathcal{L}_0$, using SGD, to obtain $f_{\mathcal{L}_0}$
for $t = 0$ **to** $T - 1$ **do**
  Compute $\Theta_{\mathcal{L}_t}(\mathcal{X}_{\mathcal{U}_t}, \mathcal{X}_{\mathcal{L}_t})$, $\Theta_{\mathcal{L}_t}(\mathcal{X}_{\mathcal{L}_t}, \mathcal{X}_{\mathcal{L}_t})$ and $\Theta_{\mathcal{L}_t}(\mathcal{X}_{\mathcal{L}_t}, \mathcal{X}_{\mathcal{L}_t})^{-1}$
  for x' **in** $\mathcal{U}_t$ **do**
    Estimate the label for x' as $y' = \operatorname{argmax} f_{\mathcal{L}_t}(x')$
    Approximate $f_{\mathcal{L}_t \cup (x', y')}$ using (11) and (12) for faster computation
    Track x^* as the point with maximal $A_{MLMOC}(x', f_{\mathcal{L}_t}, \mathcal{L}_t, \mathcal{U}_t)$ we've seen so far
  end for
  Obtain the label y^* for x^* from the oracle
  Update the labeled set $\mathcal{L}_{t+1} = \mathcal{L}_t \cup \{(x^*, y^*)\}$ and unlabeled set $\mathcal{U}_{t+1} = \mathcal{U}_t \setminus \{(x^*, y^*)\}$
  Train $f_{\mathcal{L}_t}$ on $\mathcal{L}_{t+1}$, using SGD, to obtain $f_{\mathcal{L}_{t+1}}$
end for

The dominant term for naïve SGD retraining is $\mathcal{O}(LUPE)$ whereas the proposed method using block structure is dominated by $\mathcal{O}(LUP)$ time if $U > L$, otherwise $\mathcal{O}(L^2P)$ time. If $U > L$, then the proposed method is faster by an order of $\mathcal{O}(E)$. In the other case, if $L > EU$, one can use the conjugate gradient kernel regression solvers as in Rudi et al. [27] and Gardner et al. [28] that further reduce the time complexity of our proposed method to $\mathcal{O}(\max(PL\sqrt{L}, PU\sqrt{U}))$ depending on whether $L > U$ or $L < U$, which is faster than the naïve SGD training by an order of $\mathcal{O}(UE/\sqrt{L})$ or $\mathcal{O}(LE/\sqrt{U})$.

MLMOC querying. Instead of the EMOC acquisition function (3), we define a new function more suitable for a linearized network, which we term *Most Likely Model Output Change* (MLMOC):

$$A_{MLMOC}(x', f_{\mathcal{L}}, \mathcal{L}_t, \mathcal{U}_t) := \sum_{x \in \mathcal{U}} \|f_{\mathcal{L}}(x) - f^{lin}_{\mathcal{L}^+}(x))\|_2 \tag{13}$$

where $\mathcal{L}^+ = \mathcal{L} \cup \{(x', y')\}$ with $y' = \operatorname{argmax} f_{\mathcal{L}}(x')$. We only consider the most likely pseudo-label, instead of the expectation; this saves a significant amount of computation, by a factor of the number of classes, but empirically does not hurt accuracy. We also suspect that, when $f_{\mathcal{L}}$ is already reasonably accurate, we can "trust" the most likely label more than we can trust accurate estimation of probabilities for low-probability losses, especially when training networks with L_2 loss. Algorithm 1 shows pseudo-code for the proposed algorithm.

Sequential query strategy. In the active learning literature, each query of new data points is essentially always followed by retraining the underlying model. This, however, may not be practical for the cases where annotation is quick but SGD training of a large neural network is slow. Although there has been significant effort towards batch selection strategies based on finding diverse examples to query in a batch [e.g. 4, 29, 30], which among other benefits can minimize the number of times we must retrain with SGD, they cannot fully take advantage of low-latency annotations if they are available. (This may be the case if, for instance, labeling tasks can be quickly pushed out to a pool of human labelers available for many such annotation tasks at once, or other cases where a limited number of queries are desired to avoid expense, damage to a system being measured, or so on, but those individual measurements can still be taken promptly.)

Our proposed linearized network can take advantage of sequentially added annotations without requiring SGD retraining steps. Given an empirical NTK Θ, adding another training point (x, y) requires nothing more than adding a row and column to the kernel. As a result, computing f^{lin} and following A_{MLMOC} with an additional data point (x, y) is almost instant. This approximation can utilize the information from the label y directly while batch selection strategies cannot. (After adding a substantial number of points, we will want to retrain, but it is not needed after every new data point.) We experimentally demonstrates the effectiveness of this sequential query strategy in Section 5.

4 Related Work

Two main approaches in pool-based active learning are uncertainty and representation-based methods. The general goal of the uncertainty-based methods is to query the most "informative" data points, thus, their acquisition functions vary depending on how to measure the informativeness of a data point. As simple but effective uncertainty-based querying methods, posterior probability [31, 32], entropy [1], margin sampling [33] and least confident [11], have been widely used in practice. They do not, however, take into account how a candidate data point would change a model, or how this change would interact with unseen examples. More advanced querying methods have thus been proposed as discussed in Section 2. There are also Bayesian uncertainty-based methods, particularly Bayesian Active Learning by Disagreement (BALD) [2], which measures the mutual information between a new data point and model parameters, which Gal et al. [34] propose to estimate efficiently with Monte Carlo dropout networks [35].

The goal of the representation-based methods is to query examples that are the most "representative" among the unlabeled set $\mathcal{U}$, hoping that doing well on those examples leads to doing well on the whole unseen dataset. Sener and Savarese [36] define a core set as a batch of images that minimize the distance between unlabeled and labeled images if added to the labeled pool. Kirsch et al. [30] further expand BALD to a batch setting using a greedy algorithm. Bıyık et al. [29] propose to use determinantal point processes to target diversity of the queried examples while maintaining informativeness. BADGE [4] measures the uncertainty using a gradient norm but also encourage the diversity of queried images using k-means++ seeding [37].

Other approaches include that of Yoo and Kweon [38], who employ an additional loss prediction module to an existing neural network. However, the additional loss computation module can make training unstable (as we observed in some experiments). Tran et al. [39] improves BALD by adding a variational autoencoder and auxiliary classifier generative adversarial network, from which a synthesized data point x' similar to the queried image x^* is generated.

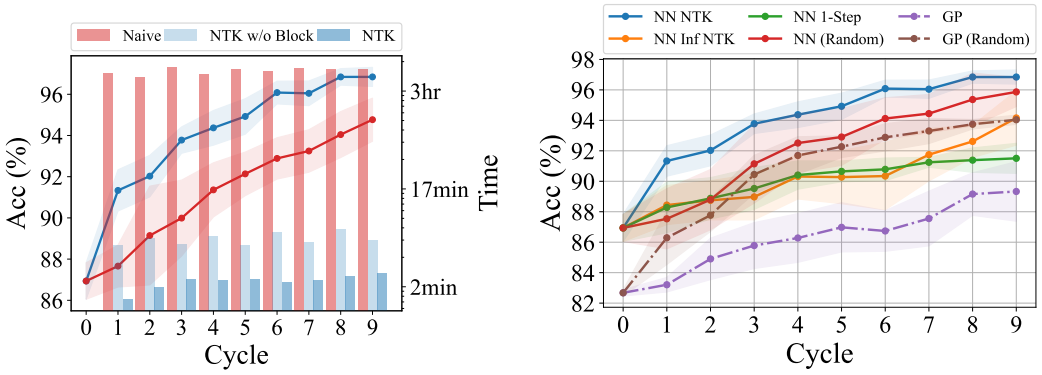
Our proposed method is different from the previous works in that it "looks one step ahead" instead of relying on the current state of a model. Perhaps the most similar proposal to ours is that of Borsos et al. [40], who propose using the infinite NTK to solve a bi-level optimization problem for semi-supervised batch active learning. Aside from the difference in our objective functions, however, using the infinite NTK to approximate a neural network in active learning does not seem to be a good choice; as the size of training set increases, the error of the approximation can be quite large. We explore the difference between infinite and empirical NTKs experimentally in Section 5.

5 Experiment Results

Datasets. To demonstrate the effectiveness of the proposed method, we provide experimental results on three benchmark datasets for classification tasks: MNIST [8], SVHN [9], CIFAR10 [10], and CIFAR100 [10]. MNIST consists of 10 hand-written digits with $60,000$ training and $10,000$ test images, at size 28×28. SVHN also consists of 10 digit numbers with $73,257$ training and $26,032$ test images, at size 32×32; it is more challenging than MNIST, containing images from pictures of house numbers, with much more variation and many distractions present. CIFAR10 contains $50,000$ training and $10,000$ test images, also 32×32 and equally split between 10 classes like `airplane`, `frog`, and `truck`. CIFAR100 is the same as the CIFAR10, except it has 100 classes; the resolution is still 32×32, and images are equally split between 100 classes. The 100 classes are grouped with 20 super-classes, but we use only the 100 sub-classes in this work.

Implementation. We implement a pipeline for the proposed method using PyTorch [41] and Jax [42]; our neural networks f are implemented in PyTorch, whereas the linearized models f^{lin} are implemented in Jax with the `neural-tangents` library [43]. We employ a ResNet18 [44] and WideResNet [45] with one or two layers and maximum width of 640, which is wide enough for a linearized neural network to be a good approximation, while still being powerful enough for these datasets. As we will see in our experiments, even for narrower networks, the proposed method outperforms random acquisition strategy by large margins.

As mentioned earlier, we use L_2 loss, rather than cross-entropy; this allows for a faster NTK approximation, as in the formulas of Section 3, rather than requiring differential equation solvers.



(a) Naïve look-ahead acquisition versus NTK approximation. Bars show runtime per cycle.

(b) MLMOC on various models, along with random-acquisition baselines.

Figure 1: Comparisons of several related approaches on MNIST.

Methods	NN NTK	NN 1-step	NN Inf NTK	GP
NN NTK	1.0000	0.0109	0.0093	0.0014
NN 1-step	0.0109	1.0000	0.6863	0.0422
NN Inf NTK	0.0093	0.6863	1.0000	0.0742
GP	0.0014	0.0422	0.0742	1.0000

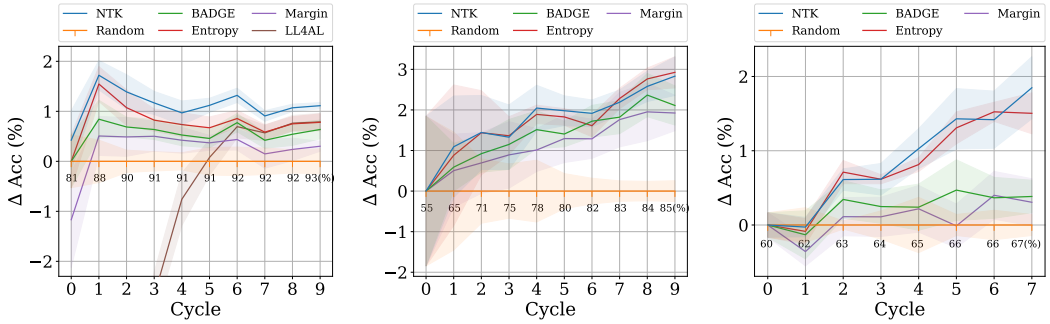
Table 1: The p-values of the post-hoc paired t-test.

Appropriately tuned L_2 loss is as effective as cross-entropy [22]. To ensure that the empirical NTK of the training data ($\mathcal{X}_\mathcal{L}$) is positive semidefinite, in batch normalization layers, we freeze the current running statistics. We implement LL4AL [38] and BADGE [4] based on their publicly available code. We provide anonymous code for the full pipeline in the supplementary material.

Query scheme. Following active learning literature, for each query step, we randomly draw a subset from an unlabeled set, and query a fixed number of examples from the subset. To build a batch, we take the points with the highest A_{MLMOC} scores, which we found to perform well without any batch diversity requirements. At each cycle, we query 20, 1 000, 1 000, and 1 000 new data points on MNIST, SVHN, CIFAR10, and CIFAR100 from the subset of 4 000, 6 000, 6 000, and 6 000 unlabeled data points, and initialize the labeled set $\mathcal{L}$ with 100, 1 000, 1 000, and 10 000 data points.

Making look-ahead acquisitions feasible. In Figure 1a, we demonstrate the NTK approximation indeed makes it feasible to use look-ahead acquisition functions. We run the MLMOC acquisition function with naïve SGD retraining (Naive) and proposed NTK approximation with (NTK) and without (NTK w/o Block) block computation in (12) on MNIST. For the naïve version, we retrain the neural network for 15 epochs using SGD. The line plots represent accuracy whereas bar plots represent wall-clock time to query data, using a NVIDIA V100 GPU. The NTK approximation is not only computationally efficient (more than 100 times faster than Naïve) but also yields significantly more accurate models. Although one might expect that Naïve is an upper bound for the NTK approximation, as it actually "looks ahead," in reality it is hard to retrain a neural network until convergence for every candidate data point; a "proper" look-ahead method should perhaps look at an ensemble of several training runs, with different learning rate schedules, etc.

Comparison of look-ahead acquisitions. Having seen that the empirical NTK effectively approximates (even outperforms) the Naïve look-ahead acquisition function, we now show the superiority of the empirical NTK over existing look-ahead methods. In Figure 1b, we provide four look-ahead methods using MLMOC acquisition function along with their corresponding random baselines (Random). NN refers to a neural network-based model whereas GP refers to a model based on Gaussian process. Instead of using empirical NTK as we propose (denoted as NN NTK, blue), Borsos et al.



(a) SVHN: 1-layer WideResNet (b) CIFAR10: 2-layer WideResNet (c) CIFAR100: ResNet18

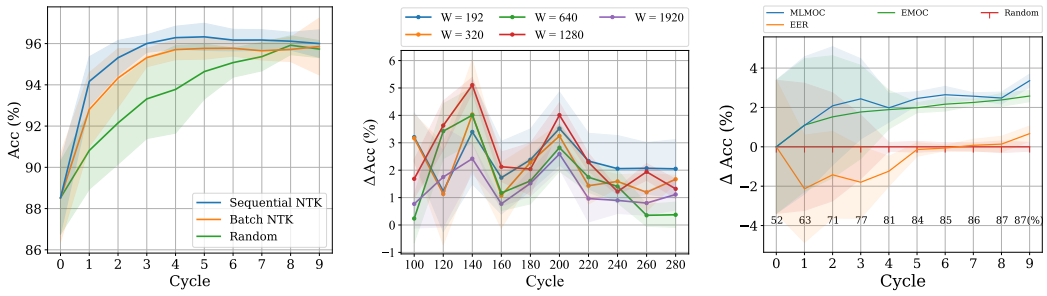
Figure 2: Comparison of the-state-of-the-art active learning methods on various benchmark datasets. Vertical axis shows difference from random acquisition, whose accuracy is shown in text.

[40] use infinite NTK (NN Inf NTK, orange). Käding et al. [14] approximate SGD retraining using only one-step SGD update (NN 1-step, green) whereas Käding et al. [15] exactly compute look-ahead acquisition using GP for regression problems. We modify their model for classification tasks, use an infinite NTK, and denote it as GP (purple). As shown in Figure 1b, ours is the only of these methods to outperform random baselines here. Neither infinite NTK nor 1-step approximation are good approximation for retraining steps, especially as more data points are added. Also, although GP [15] works well on regression tasks, it struggles on classification tasks.

To validate the difference between different look-ahead strategies, we conduct Friedman's test followed by post-hoc paired t-test on the existing look-ahead strategies in Figure 1b: NN NTK, NN 1-step, NN Inf NTK, and GP. Friedman's test is a non-parametric statistical test. The null hypothesis in our case is that the population mean of the performance of each look-ahead strategy is the same and the alternative hypothesis is that at least one population mean is different from the rest of look-ahead strategies. The p-value of the Friedman's test is 0.0045 (< 0.05). Hence, we reject the null hypothesis. As it means at least one population mean is different from the rest of look-ahead strategies, we further conduct the post-hoc paired t-test to check where the significant difference is coming from. The result of the post-hoc paired t-test is shown in Table 1. Here, the null hypothesis for each pair is that the population mean of two methods is the same. As all the p-values between ours (NN NTK) and the other methods are less than the conventional significance level (0.05), we conclude that our proposed method is significantly better than the other look-ahead strategies.

Comparison with state-of-the-art. In Figure 2, we compare the proposed NTK active learning to state-of-the-art methods – Random, Entropy, Margin sampling (Margin) [46], BADGE [4], and LL4AL [38] – on SVHN, CIFAR10, and CIFAR100 with different architectures; we provide additional results in Appendix B. For clarity, we show the difference between each method and Random. The numbers below Random line give the accuracies of Random at each cycle. As mentioned earlier, training LL4AL is often unstable due to loss computation module; in particular, it performs too poorly to show on the same plot for Figure 2b and Figure 2c. We show the instability of training LL4AL with varying learning rate in Appendix C. Each experiment is run for six different seeds; lines show the mean performance, and shading shows a 95% confidence interval for the mean. Figure 2(a) shows that the NTK method outperforms all the state-of-the-art methods throughout training on SVHN. A similar pattern is observed on CIFAR10 and CIFAR100; NTK is consistently comparable to or better than the best competitor methods, far outperforming existing look-ahead approaches.

Sequential query strategy. As mentioned in Section 3, one great advantage of the NTK approximation is that it can exploit additional annotations without needing to run SGD training for a neural network, which cannot be done in existing active learning methods for neural networks. On MNIST, Figure 3a compares sequential NTK where the oracle provides true label for every new queried data point, to batch NTK where annotation is done batch-wise as with a normal active learning setting. Sequential NTK outperforms batch NTK; especially, sequential NTK reaches more than 94% in one cycle, and in cycle 3, it has already reached 96% accuracy, where batch NTK plateaus. We expect the sequential NTK can be useful in practice when annotation is much faster than SGD training.



(a) Sequential *vs.* batch querying. (b) Performance with varying width. (c) Diff. look-ahead strategies.

Figure 3: Further comparisons of algorithm variants on MNIST.

Varying model widths. As shown in Equation (9), the NTK approximation gets closer to the dynamics of a neural network as the width of the neural network increases. But, since very wide networks are computationally expensive in practice, it is important to empirically demonstrate that the NTK approximation works even with reasonably narrow networks. To this end, we vary the maximum width of the WideResNet with one block layer we use for the experiments as shown in Figure 3b. We can observe that regardless of the maximum width, the NTK method significantly outperforms Random strategy until convergence, indicating that the NTK approximation is good enough for the purpose of active learning.

Comparison of different acquisition strategies. We compare different look-ahead acquisition functions in Figure 3c. As mentioned with Equation (13) in Section 3, MLMOC and EMOC are not significantly different. As MLMOC is computationally cheaper than EMOC proportional to the number of classes, we use MLMOC over EMOC as an acquisition function. But note that, the proposed NTK approximation is applicable to any look-ahead acquisition functions including MLMOC, EMOC and EER. Thus, we hope that our method would provide a framework for other researchers to build upon and propose various look-ahead active learning methods or enhance the look-ahead approximation to help further tackle the problem of pool-based active learning.

6 Conclusion

We propose an algorithm to make look-ahead acquisition strategies feasible for active learning with fairly general neural network architectures. We prove that the outputs of wide enough neural networks can be efficiently approximated using a linearized network using empirical neural tangent kernels. We empirically show that this approximation is at least 100 times faster than naïve computation of look-ahead strategies, while even being more accurate. Armed with this approximation, we significantly outperform previous look-ahead strategies, and achieve equal or better performance compared to the state-of-the-art methods on four widely used benchmark datasets in active learning regime. We also propose a new sequential active learning algorithm that decouples querying and SGD training for the first time.

One limitation of the NTK approximation is that it is still slower than common "myopic" active learning methods such as entropy and BADGE. Compared to our proposed method where it takes $\mathcal{O}(\max(PL\sqrt{L}, PU\sqrt{U}))$ using conjugate gradient kernel regression solvers, entropy requires $O(PU)$ time (using notations in Section 3). Improving time complexity of the NTK approximation would be an interesting future work along with devising better look-ahead acquisition functions that improve performance.

Acknowledgments and Disclosure of Funding

This research was enabled in part by support, computational resources, and services provided by the Canada CIFAR AI Chairs program, the Natural Sciences and Engineering Research Council of Canada, Advanced Research Computing at the University of British Columbia, WestGrid, Calcul Québec, and the Digital Research Alliance of Canada. We would like to specifically thank Roman Baranowski for helping us with our computational requirements.

References

[1] Dan Wang and Yi Shang. "A New Active Labeling Method for Deep Learning." *IJCNN*. 2014, pp. 112–119.

[2] Neil Houlsby, Ferenc Huszár, Zoubin Ghahramani, and Máté Lengyel. "Bayesian Active Learning for Classification and Preference Learning" (2011). arXiv: 1112.5745.

[3] Burr Settles, Mark Craven, and Soumya Ray. "Multiple-instance Active Learning." 2007, pp. 1289–1296.

[4] Jordan T Ash, Chicheng Zhang, Akshay Krishnamurthy, John Langford, and Alekh Agarwal. "Deep batch active learning by diverse, uncertain gradient lower bounds." *ICLR* (2020).

[5] Nicholas Roy and Andrew McCallum. "Toward Optimal Active Learning through Monte Carlo Estimation of Error Reduction." *ICML*. 2001.

[6] Alexander Freytag, Erik Rodner, and Joachim Denzler. "Selecting Influential Examples: Active Learning with Expected Model Output Changes." *ECCV*. 2014, pp. 562–577.

[7] Arthur Jacot, Franck Gabriel, and Clément Hongler. "Neural Tangent Kernel: Convergence and Generalization in Neural Networks." *NeurIPS*. 2018.

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

[9] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. "Reading Digits in Natural Images with Unsupervised Feature Learning." *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*. 2011.

[10] Alex Krizhevsky. "Learning Multiple Layers of Features from Tiny Images" (2009).

[11] Burr Settles. "Active Learning Literature Survey" (2009).

[12] Vu-Linh Nguyen, Sébastien Destercke, and Eyke Hüllermeier. "Epistemic Uncertainty Sampling." *ICDS*. 2019.

[13] Xiaojin Zhu, John Lafferty, and Zoubin Ghahramani. "Combining Active Learning and Semi-supervised Learning Using Gaussian Fields and Harmonic Functions." *ICMLW*. Vol. 3. 2003.

[14] Christoph Käding, Erik Rodner, Alexander Freytag, and Joachim Denzler. "Active and Continuous Exploration with Deep Neural Networks and Expected Model Output Changes" (2016).

[15] Christoph Käding, Erik Rodner, Alexander Freytag, Oliver Mothes, Björn Barz, Joachim Denzler, and Carl Zeiss AG. "Active Learning for Regression Tasks with Expected Model Output Changes." *BMVC*. 2018, p. 103.

[16] Alexander G de G Matthews, Mark Rowland, Jiri Hron, Richard E Turner, and Zoubin Ghahramani. "Gaussian Process Behaviour in Wide Deep Neural Networks." *ICLR*. 2018. arXiv: 1804.11271.

[17] Radford M Neal. "Priors for Infinite Networks." *Bayesian Learning for Neural Networks*. 1996.

[18] Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Ruslan Salakhutdinov, and Ruosong Wang. "On Exact Computation with an Infinitely Wide Neural Net." *NeurIPS*. 2019.

[19] Roman Novak, Lechao Xiao, Jiri Hron, Jaehoon Lee, Alexander A Alemi, Jascha Sohl-Dickstein, and Samuel S Schoenholz. "Neural Tangents: Fast and Easy Infinite Neural Networks in Python." *ICLR*. 2020.

[20] Greg Yang. *Tensor Programs II: Neural Tangent Kernel for Any Architecture*. 2020. arXiv: 2006.14548.

[21] Greg Yang and Etai Littwin. *Tensor Programs IIb: Architectural Universality of Neural Tangent Kernel Training Dynamics*. 2021. arXiv: 2105.03703.

[22] Like Hui and Mikhail Belkin. "Evaluation of Neural Architectures Trained with Square Loss vs Cross-entropy in Classification Tasks." *ICLR* (2021).

[23] Zhiyuan Li, Ruosong Wang, Dingli Yu, Simon S Du, Wei Hu, Ruslan Salakhutdinov, and Sanjeev Arora. "Enhanced Convolutional Neural Tangent Kernels" (2019). arXiv: 1911.00809.

[24] Eran Malach, Pritish Kamath, Emmanuel Abbe, and Nathan Srebro. "Quantifying the Benefit of Using Differentiable Learning over Tangent Kernels." *ICML*. 2021. arXiv: 2103.01210.

[25] Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. "Wide Neural Networks of Any Depth Evolve as Linear Models Under Gradient Descent." *NeurIPS*. 2019.

[26] Alexander Wei, Wei Hu, and Jacob Steinhardt. *More Than a Toy: Random Matrix Models Predict How Real-World Neural Representations Generalize*. 2022. arXiv: 2203.06176.

[27] Alessandro Rudi, Luigi Carratino, and Lorenzo Rosasco. "Falkon: An optimal large scale kernel method." 2017.

[28] Jacob Gardner, Geoff Pleiss, Kilian Q Weinberger, David Bindel, and Andrew G Wilson. "GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration." *NeurIPS*. 2018.

[29] Erdem Bıyık, Kenneth Wang, Nima Anari, and Dorsa Sadigh. "Batch Active Learning Using Determinantal Point Processes." *NeurIPS* (2019).

[30] Andreas Kirsch, Joost Van Amersfoort, and Yarin Gal. "Batchbald: Efficient and Diverse Batch Acquisition for Deep Bayesian Active Learning." 2019, pp. 7026–7037.

[31] David D Lewis and Jason Catlett. "Heterogeneous Uncertainty Sampling for Supervised Learning." *Machine learning proceedings 1994*. 1994, pp. 148–156.

[32] David D Lewis and William A Gale. "A Sequential Algorithm for Training Text Classifiers." *SIGIR*. 1994, pp. 3–12.

[33] Tobias Scheffer, Christian Decomain, and Stefan Wrobel. "Active Hidden Markov Models for Information Extraction." *ISIDA*. 2001.

[34] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. "Deep Bayesian Active Learning with Image Data." *ICML*. 2017, pp. 1183–1192.

[35] Yarin Gal and Zoubin Ghahramani. "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning." *ICML*. 2016, pp. 1050–1059.

[36] Ozan Sener and Silvio Savarese. "Active Learning for Convolutional Neural Networks: A Core-set Approach." *ICLR* (2018).

[37] David Arthur and Sergei Vassilvitskii. "K-means++: the Advantages of Careful Seeding." *SODA*. Philadelphia, PA, USA, 2007, pp. 1027–1035.

[38] Donggeun Yoo and In So Kweon. "Learning Loss for Active Learning." *CVPR*. 2019, pp. 93–102.

[39] Toan Tran, Thanh-Toan Do, Ian Reid, and Gustavo Carneiro. "Bayesian Generative Active Deep Learning." *ICML*. 2019, pp. 6295–6304.

[40] Zalán Borsos, Marco Tagliasacchi, and Andreas Krause. "Semi-supervised Batch Active Learning via Bilevel Optimization." *ICASSP*. IEEE. 2021.

[41] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." *NeurIPS*. 2019, pp. 8024–8035.

[42] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. *JAX: composable transformations of Python+NumPy programs*. Version 0.2.5. 2018. URL: http://github.com/google/jax.

[43] Roman Novak, Lechao Xiao, Jiri Hron, Jaehoon Lee, Alexander A. Alemi, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. "Neural Tangents: Fast and Easy Infinite Neural Networks in Python." *ICLR*. 2020.

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition." *CVPR*. 2016.

[45] Sergey Zagoruyko and Nikos Komodakis. "Wide Residual Networks." *BMVC* (2016).

[46] Dan Roth and Kevin Small. "Margin-based Active Learning for Structured Output Spaces." *European Conference on Machine Learning*. Springer. 2006, pp. 413–424.

[47] Leslie N Smith and Nicholay Topin. "Super-convergence: Very fast training of neural networks using large learning rates." *Artificial intelligence and machine learning for multi-domain operations applications*. 2019. arXiv: 1708.07120.

[48] Yufan Zhou, Zhenyi Wang, Jiayi Xian, Changyou Chen, and Jinhui Xu. "Meta-learning with neural tangent kernels." *NeurIPS* (2021).

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] We use only standard datasets.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] We use only standard datasets.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Proof of Theorem 3.1

In this section, we show that in the infinite width limit, warm starting a neural network will result in the same predictions as the cold started variant when trained using gradient descent for $t = \infty$. We give the proof for a feed-forward neural network, but as shown by Yang and Littwin [21], it is trivial to show that the same proof structure can be used for any other architecture.

Background We mostly use the same notation as Jacot et al. [7]. We focus on fully connected feed-forward neural networks with $L+1$ hidden layers numbered from 0 (input) to L (output), where each layer has $n_0, \ldots, n_L$ neurons. We assume that its nonlinearity is a Lipschitz, twice differentiable function $\sigma : \mathbb{R} \to \mathbb{R}$. Such a network has $P = \sum_{l=0}^{L-1} (n_l + 1)n_{l+1}$ parameters: A weight matrix $W^{(l)} \in \mathbb{R}^{n_l \times n_{l+1}}$ and a bias vector $b^{(l)} \in \mathbb{R}^{n_{l+1}}$ per layer. This network is defined as f_θ where $\theta = \cup_{l=0}^{L} \theta^l$ and $\theta^l = vec(W^{(l)}, b^{(l)})$. The function f_θ is characterized according to the definition:

$$\begin{aligned} \alpha^{(0)}(x;\theta) &= x \\ \tilde{\alpha}^{(l+1)}(x;\theta) &= \frac{1}{\sqrt{n_l}} W^{(l} \alpha^{(l)}(x;\theta) + \beta b^{(l)} \\ \alpha^{(l)}(x;\theta) &= \sigma(\tilde{\alpha}^{(l+1)}(x;\theta)) \\ f_\theta(x) &\coloneqq \tilde{\alpha}^{(L)}(x;\theta) \end{aligned} \tag{14}$$

We define the *realization function* of the network as $f_{(L)} : \mathbb{R}^P \to \mathcal{F}$, the function mapping parameters $\theta : \mathbb{R}^P$ to functions $f_\theta \in \mathcal{F}$ where $\mathcal{F}$ is the space of all neural networks of this architecture.

Assuming that the training set is defined as $\mathcal{D} \subseteq \mathbb{R}^{n_0} \times \mathbb{R}^{n_L}$, we define p^{in} to be the fixed empirical distribution of the finite input dataset $\{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)\} : \frac{1}{N} \sum_{i=0}^{N} \delta_{x_i}$. Thus, $\mathcal{F}$ is defined as all the functions $\{f : \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}\}$. On this space, we consider the seminorm $|| \cdot ||_{p^{in}}$, defined as

$$\langle f, g \rangle_{p^{in}} = \mathbb{E}_{x \sim p^{in}}[f(x)^T g(x)]. \tag{15}$$

The dual space of $\mathcal{F}$ with respect to p^{in} can be defined as $\mathcal{F}^*$. $\mathcal{F}^*$ is the set of all linear forms $\mu : \mathcal{F} \to \mathbb{R}$. We can define each element of this set as $\mu = \langle d, \cdot \rangle_{p^{in}}$ for some $d \in \mathcal{F}$.

In the process of optimizing the parameters of a neural network f, we define the cost functional as: $C(f) = \frac{1}{N} \sum_{i=1}^{N} (f(x_i) - y_i)^2$. As we assumed that p^{in} is fixed, the value of $C(f)$ only depends on the values of $f \in \mathcal{F}$ at the datapoints $(x, y) \in p^{in}$. Thus, the functional derivative of the cost functional can be viewed as a member of the dual space of $\mathcal{F}$, namely $\mathcal{F}^*$. We note by $d|_{f_0} \in \mathcal{F}$ the corresponding dual element of partial derivative of the cost functional with respect to the function f at f_0, namely: $\partial_f^{in} C|_{f_0}$. Thus, $\partial_f^{in} C|_{f_0} = \langle d|_{f_0}, \cdot \rangle_{p^{in}}$.

According to Jacot et al. [7], *Kernel Gradient* $\nabla_{\mathcal{K}} C|_{f_0} \in \mathcal{F}$ is defined as $\Phi_{\mathcal{K}} \left(\partial_f^{in} C|_{f_0} \right)$ where Φ is a map from $\mathcal{F}^*$ to $\mathcal{F}$ defined as: $[\Phi_{\mathcal{K}}(\mu)]_{i,\cdot}(x) = \langle d, \mathcal{K}_{i,\cdot}(x, \cdot) \rangle_{p^{in}}$ where $\mu = \langle d, \cdot \rangle_{p^{in}}$. Here, $\mathcal{K}$ refers to a multi-dimensional kernel which is defined as a function $\mathbb{R}^{n_0} \times \mathbb{R}^{n_0} \to \mathbb{R}^{n_L \times n_L}$ such that $\mathcal{K}(x, x') = \mathcal{K}(x', x)^\top$. They showed that when trained using gradient descent on $C \circ F_{(L)}$, the neural network function's evolution in time can be captured using *kernel gradient descent* with the corresponding Neural Tangent Kernel (NTK): $\partial_t f_{\theta(t)} = -\nabla_{\mathcal{K}} C|_{f_{\theta(t)}}$ where $\mathcal{K}$ is the corresponding Neural Tangent Kernel of $f_{\theta(t)}$. For simplicity, from here onwards, we drop the θ index from a function $f_{\theta_{(t)}}$ and show it by f_t. Since one can define a map between the time t and function $f_{\theta_{(t)}}$ when the training setting is fixed, this doesn't create any confusion. Thus, the cost functional evolves as

$$\partial_t C|_{f_t} = -\langle d|_{f_t}, \nabla_{\mathcal{K}} C|_{f_t} \rangle_{p^{in}} \tag{16}$$

Here, $-d|_{f_t} \in \mathcal{F}$ defines the training direction of function $f_{(t)}$ in the function space $\mathcal{F}$ while being trained using gradient descent (flow). As mentioned earlier, when the we train using $C \circ F_{(L)}$ loss, this training direction is the dual of $\partial_f^{in} C|_{f_t}$.

Setting A. We have C overlapping datasets, $S_1, S_2, \ldots, S_C$ such that $\forall i > j; S_j \subset S_i$ and $\forall i \in [1, C]; S_i = \{(x_1, y_1), \ldots, (x_{m_i}, y_{M_i})\}$ where M_i is the size of i'th dataset. We initialize the network

f with parameters θ_0 such that θ_0 satisfies the initialization criteria mentioned in Jacot et al. [7] (namely, LeCun initialization). We define f_0 as f with parameters θ_0. f has $L+1$ layers such that in the limit $n_0, n_1, \ldots, n_L \to \infty$ sequentially. We train f sequentially on all the sets using gradient descent with $C \circ F_L$ loss for infinite time:

$$f_0 \xrightarrow[t=\infty]{S_1} f_{S_1} \xrightarrow[t=\infty]{S_2} f_{S_{1,2}} \to \ldots \xrightarrow[t=\infty]{S_C} f_{S_{1,2,\ldots,C}}. \tag{17}$$

In the following, we first note that starting from the initialization and training the network according to (17), the sum of integrals of the training directions remains stochastically bounded (Remark A.1). Thus, based on Theorem 2 in [7], the corresponding Neural Tangent Kernel of f remains asymptotically constant (and according to their Theorem 1, it converges in probability to the limiting Neural Tangent Kernel at initialization). Next, in Theorem A.2, we prove that under this setting, the resulting function of sequential warm-start training on S_1 to S_C, is the same as the function that we get when doing simple gradient descent on only the last dataset S_C when starting from initialization (Also known as cold-start). In other words, $\forall x \in \mathbb{R}^{n_0}; f_{S_{1,\ldots,C}}(x) = f_{S_C}(x)$ where f_{S_C} is derived using $f_0 \xrightarrow[t=\infty]{S_C} f_{S_C}$.

Remark A.1. *Under Setting A, assuming that we have a function $\mathcal{T} : \mathbb{R} \to \mathbb{R}$ which gets past time since starting gradient descent as the input and outputs the index i of the dataset S_i currently being trained in the sequential training process, $\lim_{T\to\infty} \int_{t=0}^{T} ||d|_{f_t}^{S_{\mathcal{T}(t)}}||_{p^{in}} dt$ is stochastically bounded.*

Proof. We start by deriving the Gâteaux derivative of the cost functional $C(f) = \frac{1}{N}\sum_{i=1}^{N} ||f(x) - y||_2^2$:

$$\partial_f^{in} C|_{f_t}(f) = \frac{2}{N}\sum_{i=1}^{N} f(x_i)^\top \left(f_t(x_i) - y_i\right) \tag{18}$$

This shows the amount of change in $C(f_t)$ when f_t is moved towards f by an infinitesimal t. As mentioned before, this functional derivative only depends on the values of f on the datapoint in p^{in}. Thus, it's in the dual space of $\mathcal{F}$, noted by $\mathcal{F}^*$ and we can write it as $\langle d|_{f_t}, f\rangle_{p^{in}}$. We're interested in deriving the closed form of $d|f_t$ in this inner product.

$$\langle d|_{f_t}, f\rangle_{p^{in}} = \frac{1}{N}\sum_{i=1}^{N} d|_{f_t}(x_i)^\top f(x_i) = \frac{2}{N}\sum_{i=1}^{N} f(x_i)^\top \left(f_t(x_i) - y_i\right) \tag{19}$$

This implies

$$\forall (x_i, y_i) \in p^{in};\ d|_{f_t}(x_i) = \frac{1}{2}\left(f_t(x_i) - y_i\right). \tag{20}$$

If we define $f^*(x)$ as the function that maps each datapoint to its label on p^{in} and is arbitrary elsewhere, $d|_{f_t}(x) = \frac{1}{2}\left(f_t(x) - f^*(x)\right)$ on p^{in}. Accordingly, when we train using only a portion of the data $S_i \subseteq p^{in}$ such that $\mathcal{X}_{S_i} = \{x : (x, y) \in S_i\}$, but the cost functional still operates on whole of p^{in}, we have that:

$$d|_{f_t}^{S_i}(x) = \frac{I(x \in \mathcal{X}_{S_i})}{2}\left(f_t(x) - f_{S_i}^*(x)\right) \tag{21}$$

where $f_{S_i}^*$ is defined on S_i as f^* is on p^{in}. To analyze $\int_{t=0}^{T} ||d|_{f_t}^{S_i}||_{p^{in}} dt$, we first derive $||d|_{f_t}^{S_i}||_{p^{in}}$:

$$||d|_{f_t}^{S_i}||_{p^{in}} = \mathbb{E}_{x\sim p^{in}}\left[\left(d|_{f_t}^{S_i}(x)\right)^2\right] = \frac{1}{4N}\sum_{x_j \in \mathcal{X}_{S_i}} ||f_t(x_j) - f_{S_i}^*(x_j)||_2^2 \tag{22}$$

As Jacot et al. [7] (Section 5) mentioned, as t grows, this norm is strictly decreasing and thus the integral is bounded. Moreover, as we're following the direction of gradient flow in the functional space, $\exists t \geq 0\ f_t(x_j) = f_t^*(x_j)$ for all datapoints x_j in S_i. Thus, $\lim_{T\to\infty} \int_{t=0}^{T} ||d|_{f_t}^{S_i}||_{p^{in}} dt$ is also stochastically bounded. We can apply the same structure for the case where we start from f_{S_i} and perform gradient flow towards $f_{S_{i+1}}$, showing that the integral in the infinite time limit is also stochastically bounded. Thus, it's straightforward to show that using induction, when training the neural network sequentially on $S_1, S_2, \ldots, S_C$ for sequentially infinite time, the integrals of training directions remain stochastically bounded. □

Theorem A.2. *Under Setting A, the following equality holds:*

$$\forall x \in \mathbb{R}^{n_0} \quad f_{S_{1,\dots,C}}(x) = f_{S_C}(x) \tag{23}$$

Proof. As Remark A.1 showed, when training a neural network on multiple overlapping datasets under Setting A, the training direction remains stochastically bounded. Thus, as the width of the layers of the network tend to infinite sequentially, we can use Jacot et al. [7]'s Theorem 2 to show that the NTK $\mathcal{K}$ also remains constant during training in this setting. Accordingly, when trained only on a dataset S_i, the neural network function f evolves as

$$\begin{aligned}\partial_t f_t(x) &= -\nabla_{\mathcal{K}} C|_{f_t}^{S_i}(x) = -\Phi_{\mathcal{K}}\left(\partial_f^{S_i} C|_{f_t}\right) = -\frac{1}{2N}\sum_{x_j\in S_i}\mathcal{K}(x,x_j)(f_t(x_j)-f^*_{S_i}(x_j)) \\ &= \underbrace{\frac{1}{2N}\sum_{x_j\in S_i}\mathcal{K}(x,x_j)f_t(x_j)}_{\text{function of } f_t(x),\, x \text{ at } t} - \underbrace{\frac{1}{2N}\sum_{x_j\in S_i}\mathcal{K}(x,x_j)f^*(x_j)}_{\text{function of } x \text{ at } t}\end{aligned} \tag{24}$$

where $\mathcal{K}$ is the corresponding NTK of f_t, which remains constant during training. If we look closely, we witness that this is a system of ODEs, whose solution for the finite dataset S_i from initialization f_0 can be written as

$$f_t(\mathcal{X}_{S_i}) = f^*_{S_i}(\mathcal{X}) + e^{-t\mathcal{K}_{S_i}}\left(f_0(\mathcal{X}_{S_i}) - f^*_{S_i}(\mathcal{X}_{S_i})\right). \tag{25}$$

where $\mathcal{K}_{S_i} = \mathcal{K}(\mathcal{X}_{S_i}, \mathcal{X}_{S_i})$. As $\Phi_{\mathcal{K}}(\cdot)$ helps us generalize the values of kernel gradient (and consecutively f_t, as it evolves according to kernel gradient) to values x outside the dataset S_i (and also p^{in}). Now that we have derived the closed form solution of the outputs of f_t on X_{S_i}, we can also derive the outputs of f_t on any arbitrary x by taking the integral of (24) and replacing f_t according to (25):

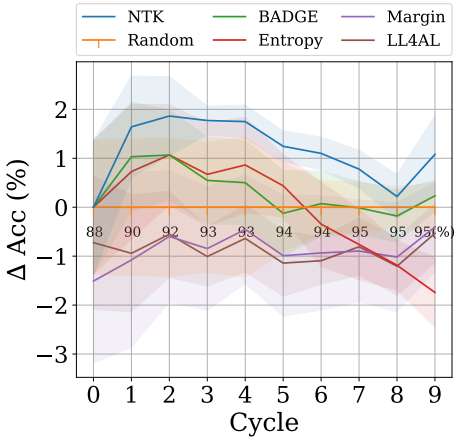
$$\begin{aligned}f_t(x) &= \frac{1}{2N}\sum_{x_j\in S_i}\mathcal{K}(x,x_j)\left[\int_{t'=0}^{t}\left(f_{t',z}(x_j) - f^*_{S_i,z}(x_j)\right)dt'\right]_z \\ &= f_0(x) + \mathcal{K}(x,\mathcal{X}_{S_i})\mathcal{K}_{S_i}^{-1}\left(I - e^{-t\mathcal{K}_{S_i}}\right)\left(f^*_{S_i}(\mathcal{X}_{S_i}) - f_0(\mathcal{X}_{S_i})\right)\end{aligned} \tag{26}$$

where $f_{\cdot,z}$ shows the zth index of output of f and $[\cdot]_z$ shows the stacked vector for different values of z, such that z is an integer in $[1 - n_L]$. Starting from initialization, we can use this to characterize the outputs of the trained neural network on S_i using gradient flow for time t. We're interested in starting from f_{S_i} and training on S_{i+1} for infinite time, according to the same loss function defined on p^{in}. For the sake of having more clear notations, we denote this function as $f_{S_i\to S_{i+1}}$ (defiend as $f_{S_i,S_{i+1}}$ in the statement that we are proving).

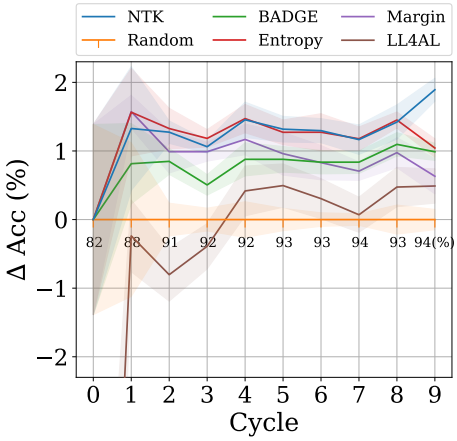
$$\begin{aligned}f_{S_i\to S_{i+1}}(x) &= f_{S_i}(x) + \mathcal{K}(x,\mathcal{X}_{S_{i+1}})\mathcal{K}_{S_{i+1}}^{-1}\left(f^*_{S_{i+1}}(\mathcal{X}_{S_i}) - f_{S_i}(\mathcal{X}_{S_{i+1}})\right) \\ &= \mathcal{K}(x,\mathcal{X}_{S_{i+1}})\mathcal{K}_{S_{i+1}}^{-1}f^*_{S_{i+1}}(\mathcal{X}_{S_{i+1}}) \\ &\quad + \underbrace{\left(\mathcal{K}(x,\mathcal{X}_{S_1}) - \mathcal{K}(x,\mathcal{X}_{S_{i+1}})\mathcal{K}_{S_{i+1}}^{-1}\mathcal{K}(\mathcal{X}_{S_{i+1}},\mathcal{X}_{S_i})\right)}_{A}\mathcal{K}_{S_i}^{-1}\left(f^*_{S_i}(\mathcal{X}_{S_i}) - f_0(\mathcal{X}_{S_i})\right) \\ &\quad + f_0(x) - \mathcal{K}(x,\mathcal{X}_{S_{i+1}})\mathcal{K}_{S_{i+1}}^{-1}f_0(\mathcal{X}_{S_{i+1}}) \\ &= f_{S_{i+1}}(x) + A\mathcal{K}_{S_i}^{-1}\left(f^*_{S_i}(\mathcal{X}_{S_i}) - f_0(\mathcal{X}_{S_i})\right)\end{aligned} \tag{27}$$

If we look closely, A can be written as:

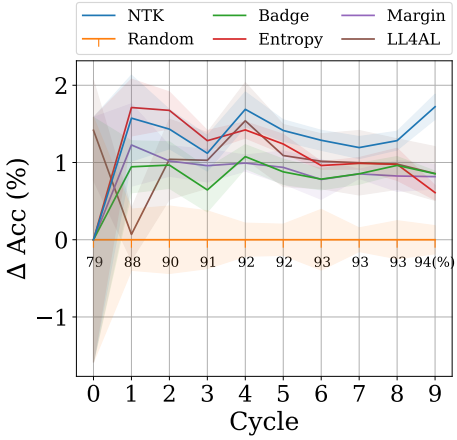
$$\begin{aligned}A &= \mathcal{K}(x,\mathcal{X}_{S_{i+1}})\begin{bmatrix} I_{M_i} \\ O_{(M_{i+1}-M_i)\times M_i}\end{bmatrix} - \mathcal{K}(x,\mathcal{X}_{S_{i+1}})\mathcal{K}_{S_{i+1}}^{-1}\mathcal{K}(\mathcal{X}_{S_{i+1}},\mathcal{X}_{S_i}) \\ &= \mathcal{K}(x,\mathcal{X}_{S_{i+}})\left(\begin{bmatrix} I_{M_i} \\ O_{(M_{i+1}-M_i)\times M_i}\end{bmatrix} - \mathcal{K}_{S_{i+1}}^{-1}\mathcal{K}(\mathcal{X}_{S_{i+1}},\mathcal{X}_{S_i})\right) \\ &= \mathcal{K}(x,\mathcal{X}_{S_{i+1}})\left(\begin{bmatrix} I_{M_i} \\ O_{(M_{i+1}-M_i)\times M_i}\end{bmatrix} - \mathcal{K}_{S_{i+1}}^{-1}\mathcal{K}(\mathcal{X}_{S_{i+1}},\mathcal{X}_{S_{i+1}})\begin{bmatrix} I_{M_i} \\ O_{(M_{i+1}-M_i)\times M_i}\end{bmatrix}\right) \\ &= \mathcal{K}(x,\mathcal{X}_{S_{i+1}}) \times 0 = 0\end{aligned} \tag{28}$$



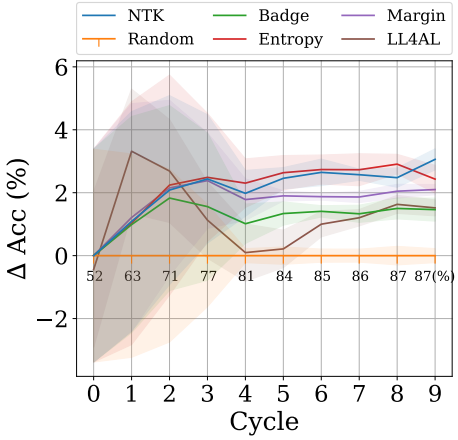
(a) MNIST: 1-layer WideResNet



(b) SVHN: 2-layer WideResNet



(c) SVHN: ResNet18



(d) CIFAR10: ResNet18

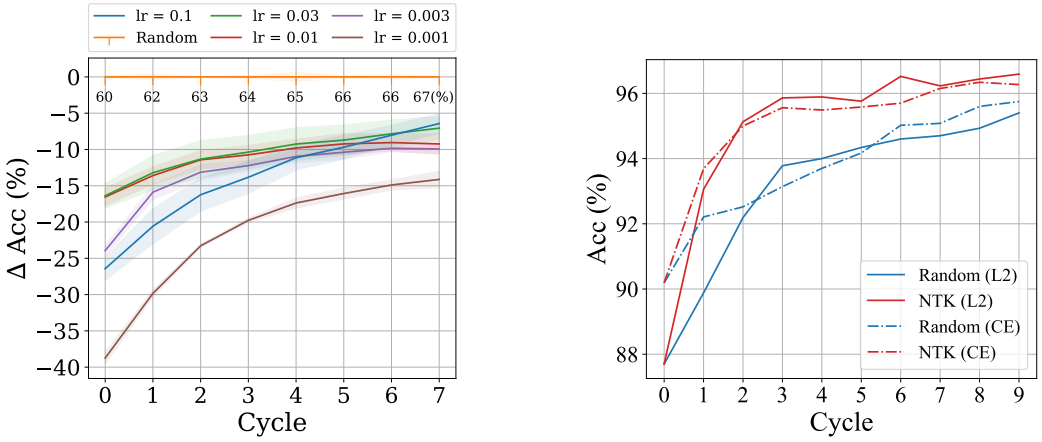
Figure 4: Comparison of the-state-of-the-art active learning methods on various benchmark datasets. Vertical axis shows difference from random acquisition, whose accuracy is shown in text.

where M_i denotes the number of datapoints in S_i. Note that (28) doesn't depend on the indices of S_i and S_{i+1}. Rather, it just relies on $S_i \subseteq S_{i+1}$, which is the case in Setting A for any consecutive batches. Thus, for any arbitrary x, $f_{S_i \to S_{i+1}}(x) = f_{S_{i+1}}(x)$. Using basic forward induction, starting from f_{S_i}, we can prove that $f_{S_1 \to S_2 \to \cdots \to S_C}(x) = f_{S_C}(x)$ for any arbitrary x. The proof is complete. □

B Additional Comparison with State-of-the-art Methods

In Figure 2, we demonstrate that our proposed method is equal or better compared to other state-of-the-art methods on various benchmark datasets in active learning, with different architectures. In this section, we provide additional experiment results as shown in Figure 2. Among all the combinations of the datasets we use (MNIST, SVHN, CIFAR10 and CIFAR100), and architectures (1-layer WideResNet, 2-layer WideResNet, and ResNet18), we do not provide results on MNIST with 2-layer WideResNet and ResNet18 since 1-layer WideResNet is large enough for MNIST. Also, we do not provide the results with 1-layer WideResNet on CIFAR10 since 1-layer WideResNet is too shallow for complicate data like CIFAR10 and 100. Similarly, according to our experiments, WideResNet with 1 or 2-layers are too shallow for CIFAR 100, unlike ResNet18.

As a result, in Figure 4, we provide the experiment results of all the other combinations. We visualize in the same way as we do for Figure 2 where we plot the difference between each method and random acquisition function at each cycle. Figure 4 shows that the proposed NTK approximation is equal or



(a) Varying LL4AL's learning rate (CIFAR100).

(b) L_2 and cross-entropy losses on MNIST.

Figure 5: Additional comparisons.

better than the other state-of-the-art methods; especially, NTK is better than any other methods for Figure 4a and Figure 4c, and comparable to entropy acquisition function for Figure 4b and Figure 4d.

C Instability of LL4AL

In Figure 2, we do not include LL4AL [38] performance on CIFAR10 or CIFAR100; its performance is too poor to show on the same plots. To validate this was not due to a simple poor hyperparameter choice, we show the performance of LL4AL on CIFAR100 with varying learning rates in Figure 5a. We run each experiment 3 times with the maximum learning rates of $\{0.3, 0.1, 0.03, 0.01, 0.003, 0.001\}$ for the 1cycle learning rate policy [47], which we use for all the other methods. When the maximum learning rate is 0.3, the gradient blows up and the model does not learn anything.

Figure 5a shows none of the learning rates are comparable with random acquisitions (trained with the maximum learning rate of 1.0), although it has been reported that LL4AL performs better than the random acquisition function. We conjecture this is because, if we train with random acquisitions using a learning rate that "works" for LL4AL, LL4AL performs better. On the other hand, when the learning rate is tuned to maximize the performance of each method, random acquisition can perform better.

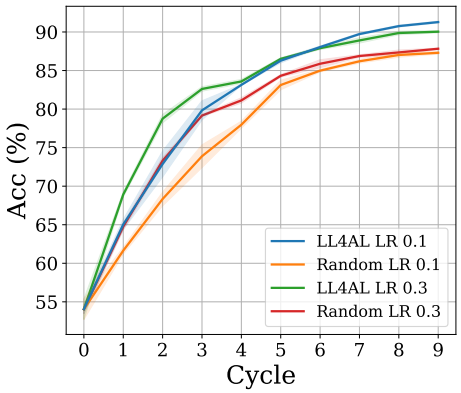
We added the figures Figure 6 based on the results shared in this GitHub repository which is a direct implementation of the LL4AL method. We further elaborate on each plot below:

CIFAR10, ResNet18 (Figure 6a): We observe that although when using the learning rate of 0.1 LL4AL's performance is 4% better than random (91.3% vs 87.3%), when the learning rate is tuned in favor of random (0.3 LR instead of 0.1) and use that for LL4AL too, the difference shrinks down to 1.2% (90.0% vs 87.8%). Moreover, using larger learning rates like 1.0 would result in LL4AL to diverge while random still gets descent results.

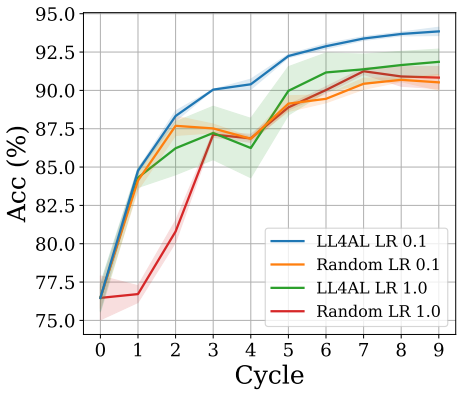
SVHN, ResNet18 (Figure 6b): Again, the difference is 3.3% (93.8% vs 90.5%) when LR is tuned in favor of LL4AL, but drops to 1.1% (91.9% vs 90.8%) when using a LR of 1.0. This time however, when LR is fully tuned in favor of random (2.0), LL4AL diverges.

CIFAR10, 1-block WideResNet (Figure 6c): The 2.6% difference when using 0.1 LR (82.2% vs 79.6%) drops to -1.8% (78.8% vs 80.6%) when LR is tuned in favor of random (0.3). We remind that as we're using one block layer here, the LL4AL's performance drop is much more noticeable to the point that it performs worse than random when LR is tuned in favor of random!

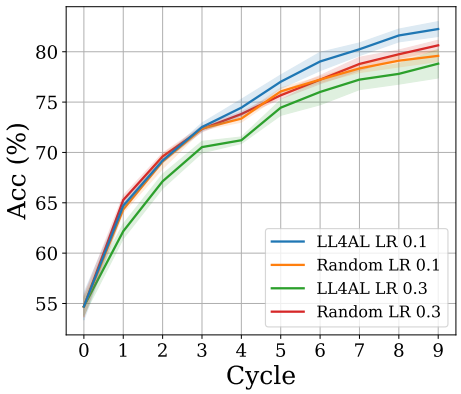
SVHN, 1-block WideResNet (Figure 6d): The 2% difference when using 0.1 LR (95% vs 93%) drops to -1.4% (92.4% vs 93.8%) when LR is tuned in favour of random (0.5).



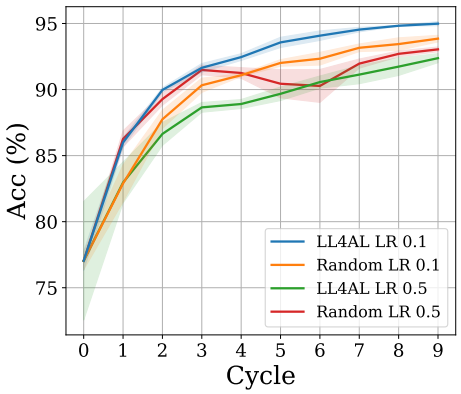
(a) Varying LL4AL's learning rate (CIFAR10 - ResNet18).



(b) Varying LL4AL's learning rate (SVHN - ResNet18).



(c) Varying LL4AL's learning rate (Cifar10 - 1-layer WideResNet).



(d) Varying LL4AL's learning rate (SVHN - 1-layer WideResNet).

Figure 6: Additional experiments on instability/poor performance of LL4AL on large learning rates.

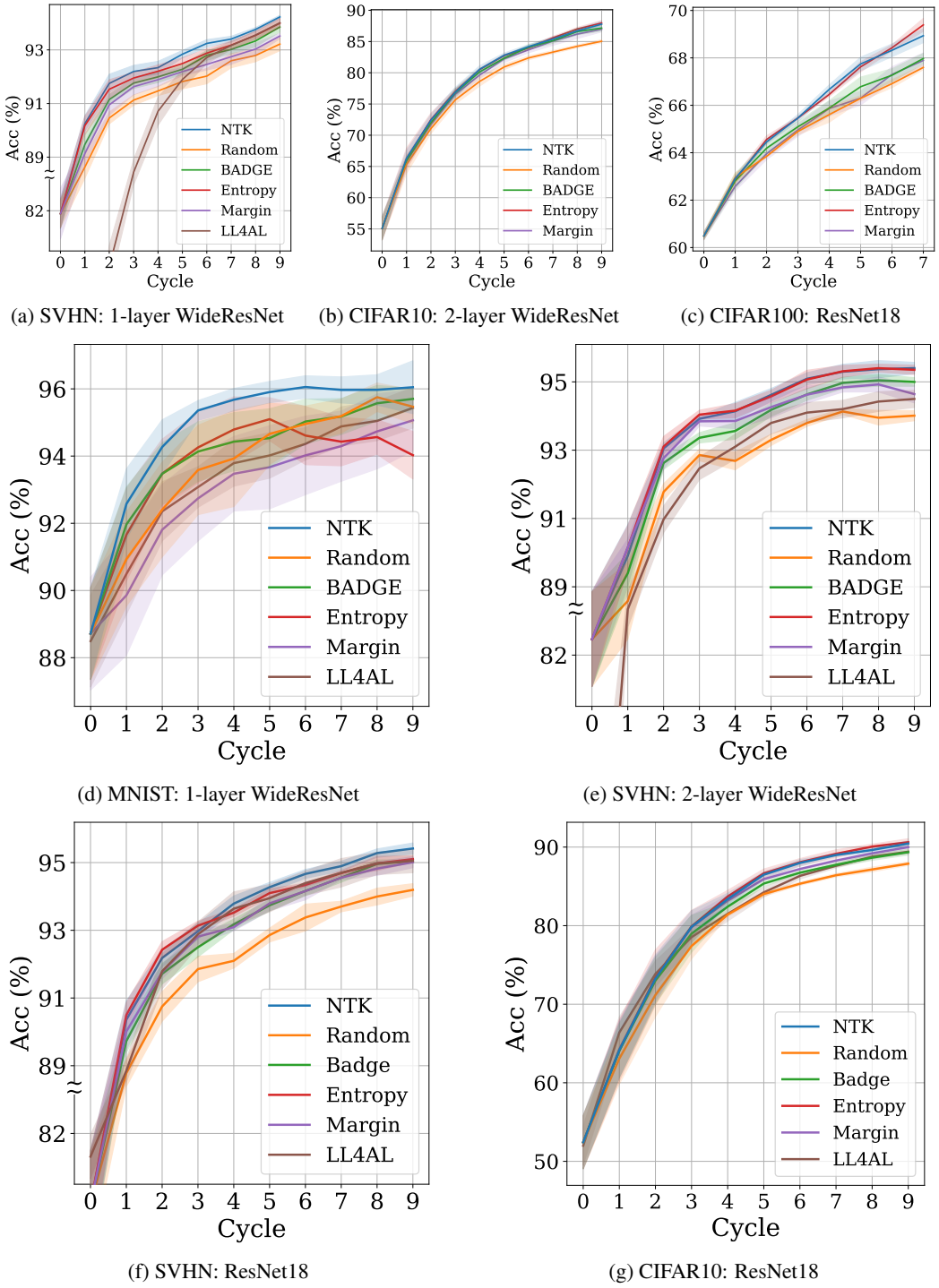
D Comparison of L_2 and Cross-entropy Loss

We use L_2 loss instead of cross-entropy loss to train a classifier. As mentioned in the main paper, Hui and Belkin [22] empirically show L_2 loss is just as effective as cross-entropy loss for various classification tasks in computer vision and natural language processing. Because of this, previous works that exploit a linearized network using empirical NTKs also use L_2 loss as a replacement for cross-entropy [25, 48].

To further demonstrate that L_2 loss is indeed as effective as cross-entropy in active learning, we provide experimental results with L_2 and cross-entropy loss (CE) for both random and the proposed NTK acquisition functions in Figure 5b. Although cross-entropy starts with a higher accuracy, L_2 quickly catches up. Overall, the difference between L_2 and cross-entropy is not significant for either random or NTK acquisition functions.

E Conventional Format for Comparison of the Proposed Method vs. SOTA

We have provided the conventional accuracy-based plots for the experimental results of comparing our proposed method with other state-of-the-art methods. We think the ΔAcc based plots are better at letting readers observe the differences between performances of each method on each task. However, for the sake of completeness and in case one finds the plots in the main paper confusing, we have provided the conventional plots in Figure 7.



(a) SVHN: 1-layer WideResNet (b) CIFAR10: 2-layer WideResNet (c) CIFAR100: ResNet18

(d) MNIST: 1-layer WideResNet (e) SVHN: 2-layer WideResNet

(f) SVHN: ResNet18 (g) CIFAR10: ResNet18

Figure 7: Comparison of the-state-of-the-art active learning methods on various benchmark datasets. Vertical axis shows attained accuracy of each acquisition method.